SAT-1.405B-1-MIP-05

ACCELERATING COMPUTATIONS ON AN ANDROID MOBILE DEVICE

Tzvetomir Vassilev, Prof. PhD
Department of Informatics,
“Angel Kanchev” University of Ruse,
Phone: +359 82 888475
E-mail: TVassilev@uni-ruse.bg

***Abstract**: This paper describes a technique for accelerating the computations on a mobile device (smartphone or tablet) using parallel computing on a multicore CPU. The paper addresses a particular example of a mass-spring cloth model for garment simulation. The simulation starts from flat garment pattern meshes positioned around a 3D human body, then seaming forces are applied on the edges of the panels until the garment is seamed and several cloth draping steps are performed in the end. The cloth-body collision detection and response algorithm is based on image-space interference tests and the cloth-cloth collision detection uses a recursive parallel algorithm on the CPU. As the results section shows the average time of dressing a virtual body with a garment on a modern smart phone is 2 seconds.*

***Keywords:** Mobile computing, Parallel computations, Cloth Simulation*

INTRODUCTION

Mobile computing has developed enormously in the last two decades. Almost everyone nowadays has a multicore CPU smartphone with Android or iOS operating system and a mobile graphics processing unit (GPU) than can be utilized to accelerate computations even further. With the development of the cloud technologies mobile computing has been extended to a new mobile cloud computing paradigm, where cloud services can be used as an additional computational power or just data storage [1, 2].

This paper proposes a technique for accelerating a virtual try-on garment simulation on a mobile device. The system has to solve several tasks. The first is the cloth simulation. We use a mass-spring model described below, which is very suitable for parallelization. Collision detection is the bottleneck in many cases of dynamic simulation. In a virtual try-on (VTO) system it has two sides: cloth-body and cloth-cloth collision detection and response. The last task is stitching the garment panels together. This is achieved via forces applied on the pattern seaming lines and the simulation is performed until the garment panels are seamed.

The rest of the paper is organized as follows. The next section reviews existing approaches for cloth modelling with different techniques for acceleration. Section 3 describes the implementation of the virtual try on system on a mobile device with a GPU that supports only OpenGL ES2. Section 4 presents and discusses some results and the last section concludes the paper.

BACKGROUND

Previous work in cloth simulation

Many computer graphics researchers have tackled the physically based cloth modelling problem for the last three decades. Terzopoulos et al. [3] viewed cloth simulation as a problem of deformable surfaces and used the finite element method and energy minimisation techniques. Provot [4] proposed a mass-spring model to describe rigid cloth behaviour, which is much faster than the techniques described above. Its main disadvantage is the unrestricted elasticity of ideal springs. In order to overcome this problem he applied a position modification algorithm to the ends of the over-elongated springs. Later Vassilev et al. [5] used a velocity modification approach to solve the super-elasticity problem.

Mass-spring model of cloth

The cloth model, utilised in this work, is based on the method described by Vassilev et al. [5]. The elastic model of cloth is a rectangular topology mesh of *m*□*n* mass points, linked to each other by massless springs of natural length greater than zero. There are three different types of springs: stretch, shear, and bending, which implement resistance to stretching, shearing and bending.

Let $\mathbf{p}_i(t)$, $\mathbf{v}_i(t)$, $\mathbf{a}_i(t)$, where i=1,…, m x n, be correspondingly the positions, velocities, and accelerations of the i-th mass point at time t. The system is governed by Newton's basic law:

$$\mathbf{a}_i = \mathbf{f}_i \,/\, mass_i, \tag{1}$$

where $mass_i$ is the mass of point $\mathbf{p}_i$ and $\mathbf{f}_i$ is the sum of all forces applied at that point. The force $\mathbf{f}_i$ can be divided in two categories.

The internal forces are a result of the tensions of the springs. The overall internal force applied at the point $\mathbf{p}_i$ is a sum of the forces caused by all springs linking this point to its neighbours:

$$f_{int}(p_i) = -\sum_l k_{il} \Delta\mathbf{l}_{ik} \, , \tag{2}$$

where k_{il} is the stiffness of the spring linking $\mathbf{p}_j$ and $\mathbf{p}_l$ and $\Delta\mathbf{l}_{il}$ is the elongation of the same spring related to its natural length.

The external forces in this cloth simulation are three types: viscous damping ($-c\,\mathbf{v}_i(t)$), gravity and seaming forces applied to the edges to be stitched together.

All the above equations make it possible to compute the force $\mathbf{f}_i(t)$ applied on cloth vertex $\mathbf{p}_i$ at any time t. The fundamental equations of Newtonian dynamics can be integrated over time by a simple Euler, Verlet or Runge-Kutta method [5].

Collision detection

Collision detection (CD) and response are the bottleneck of simulation algorithms especially when dynamically changing curved surfaces are used. Most algorithms for detecting collisions between cloth and other objects in the scene are based on geometrical object-space (OS) interference tests. Some apply a prohibitive energy field around the colliding objects [3], but most of them use geometric calculations to detect penetration between a cloth particle and a triangle of the object together with methods that reduce the number of checks.

Some common approaches use recursive subdivision with bounding box hierarchy [7, 8]. Objects are grouped hierarchically according to proximity rules, flexible surfaces are subdivided in patches and a bounding box is computed for each object or patch. The collision detection is then performed by testing for intersections of bounding boxes. Other techniques exploit proximity tracking [9] or curvature computation [7] to reduce the large number of collision checks, excluding objects or parts which are impossible to collide.

Other approaches apply image-space tests [10, 11] to detect collisions. These algorithms utilise the graphics hardware to render the scene and then use the depth map of the rendered image to perform checks for collision between objects. In this way the 3D problem is reduced to 2.5D. Vassilev et al. [5] applied a similar technique for detecting collisions between cloth and body when dressing virtual actors. They created depth, normal and velocity maps using two orthographic cameras that were placed at the centre of the front and the back face of the body's bounding box. The depth map was used for detecting collisions, while the normal and velocity maps were employed for collision response. Vassilev & Spanlang [12] implemented this approach entirely on GPU. They also proposed a method for collision detection and response between layers of cloth utilizing several depth and normal maps stored in 3D textures running on the GPU.

Cloth simulation on a mobile device

Our extensive search could not find much work in the field of cloth simulation on mobile devices. Jeon et al. [13] proposed an implementation of cloth simulation on a mobile device, which was accelerated with the help of the mobile GPU. They used a mass-spring system and a vertex shader with transform feedback (TFB) buffers, which available in OpenGL ES 3.0. However, they did not offer any solution for a device supporting only OpenGL ES 2.0, on which TFB is not available.

This paper proposes a solution for virtual try-on on a mobile device supporting only OpenGL ES 2.0.

PROPOSED APPROACH

The capabilities of a GPU supporting OpenGL ES 2.0 are quite limited for general purpose computations. Transform feedback and compute shaders are not supported in OpenGL ES 2.0, so the only way to make computations is to use the old well known render-to-texture technique. However, rendering to floating point textures is not possible either and in the best case one can set as a rendering target a RGBA texture with 8-bit unsigned integers per colour at maximum. This makes it impossible to utilize the GPU for cloth simulation, however it can be used to speed up the collision detection, as described below.

Cloth simulation

In order to increase the performance the main simulation algorithms are written in C++ using Native Development Toolkit (NDK). The NDK gcc compiler supports the OpenMP application programming interface for parallel programming. A single for loop can be parallelized using the compiler directive "#pragma omp parallel for".

One integration step of the proposed cloth simulation on the CPU can be described with the following pseudo code:

Algorithm 1: Cloth simulation on multicore CPU

```
Parallelize For each spring
  Compute internal forces
  Add forces to 2 end mass points
Endfor
Parallelize For each mass point
  Add external forces
Endfor
Parallelize For each joint
  Add joint forces
Endfor
Parallelize For each mass point
  Compute velocity
  Do collision detection and response
Endfor
Parallelize For each spring
  Correct velocities for over-elongated springs
Endfor
Parallelize For each mass point
  Compute new positions
Endfor
```

There are six main for loops suitable for parallelization. The first one is for each spring and computes the internal forces due to the stiffness of the springs. The second loop goes for each mass and adds the viscous damping and gravity. As mentioned earlier another type of external forces used in our simulation is the seaming force applied to assemble the garment from its pattern pieces. Each seaming line between two patches is composed of "joints" and each joint connects two mass points or a mass point and an edge (the line connecting two mass-points). So, the third loop of the pseudo code is for each joint and adds joint forces to connected masses. The next loop computes the velocity of each mass, checks for collisions, as described below, and modifies velocities to respond to the detected collision. Loop 5 again goes for each spring and checks if the spring elongation is above a certain threshold. If this is the case then velocities of the masses, connected by the spring, are modified as described by Vassilev et al. [5]. The last loop again goes for each mass point and computes the new positions.

SIMD instruction for accelerating computations

Most current mobile devices are based either on ARM or Intel x86 processors. Both CPU families support SIMD instructions called NEON on ARM and MMS on x86. The mass-spring cloth simulation uses quite a few operations on velocities and position which are in fact 3D vectors. So all operations on vectors (like addition, subtraction, multiplication, division, dot and vector product) can be implemented using these instructions, which speed up the computations further. There is no need to use assembly language as most compilers support the so called intrinsic operations implemented as macros or functions.

Cloth-body collision detection

The mobile GPU can be used to accelerate collision detection as described in [5, 12]. Two orthographic cameras are placed in front and in the back of the human body and render front and back depth and normal maps. The depth maps are used for detecting collisions, while the normal map, in which the XYZ coordinates of the normal vectors are encoded as RGB colours, are used for collision response. In addition, to reduce the number of texture units, the front and back maps are placed in a single texture.

Cloth-cloth collision detection

The testing for cloth-cloth collisions is performed only between different garment pieces, as self-collisions in a single patch is very unlikely to occur in a virtual try-on simulation. The implemented technique is similar to the bounding box hierarchy one described in [8] with some modifications. The algorithm checks for collisions between every two cloth patches. The function for collision detection between two cloth pieces is recursive. First it builds the axis aligned bounding boxes of the two patches and checks if they overlap. If this is true, every patch is subdivided in four sub-patches and the function is recursively called to check for collisions between each pair of patches. In order to accelerate the algorithm exploiting the multicore CPU a new parallel task is created for each recursive call of the function using #pragma omp task. Once collision is detected between two cloth faces, repulsive forces are applied to prevent penetration.

RESULTS

The system was implemented using Android SDK and Native Development Toolkit (NDK). The interface is written in Java and the main simulation algorithms are in C++ for speed.

In order to check performance, the algorithms were also implemented on a laptop PC with Windows 8.1. Tests were performed on the following configurations:

1) Windows laptop with Intel Core i7-4702MQ quad core (with 8 threads) at 2.2 GHz.

2) Android mobile device with a quad core ARM Cortex-A17 1.7 GHz CPU.

3) Android smart phone with ARM Cortex A7 quad core 1.3 GHz CPU with Mali 400 GPU supporting OpenGL ES 2.0.

Figure 1 shows the results of simulating a pair of jeans on a virtual female body in four test cases: laptop CPU without and with parallelization, mobile device (MD) CPU without and with parallelization. The time was measured in milliseconds for a single iteration for different numbers of cloth vertices. In this particular case cloth-cloth collision detection was turned off as it is not likely to happen for a single garment. The simulation on MD CPU with parallelization is quite fast and it is faster than PC CPU without parallel threads on the CPU. The simulation times on a smart mobile phone are not shown, but they are about two times slower than on the MD with ARM Cortex A17.

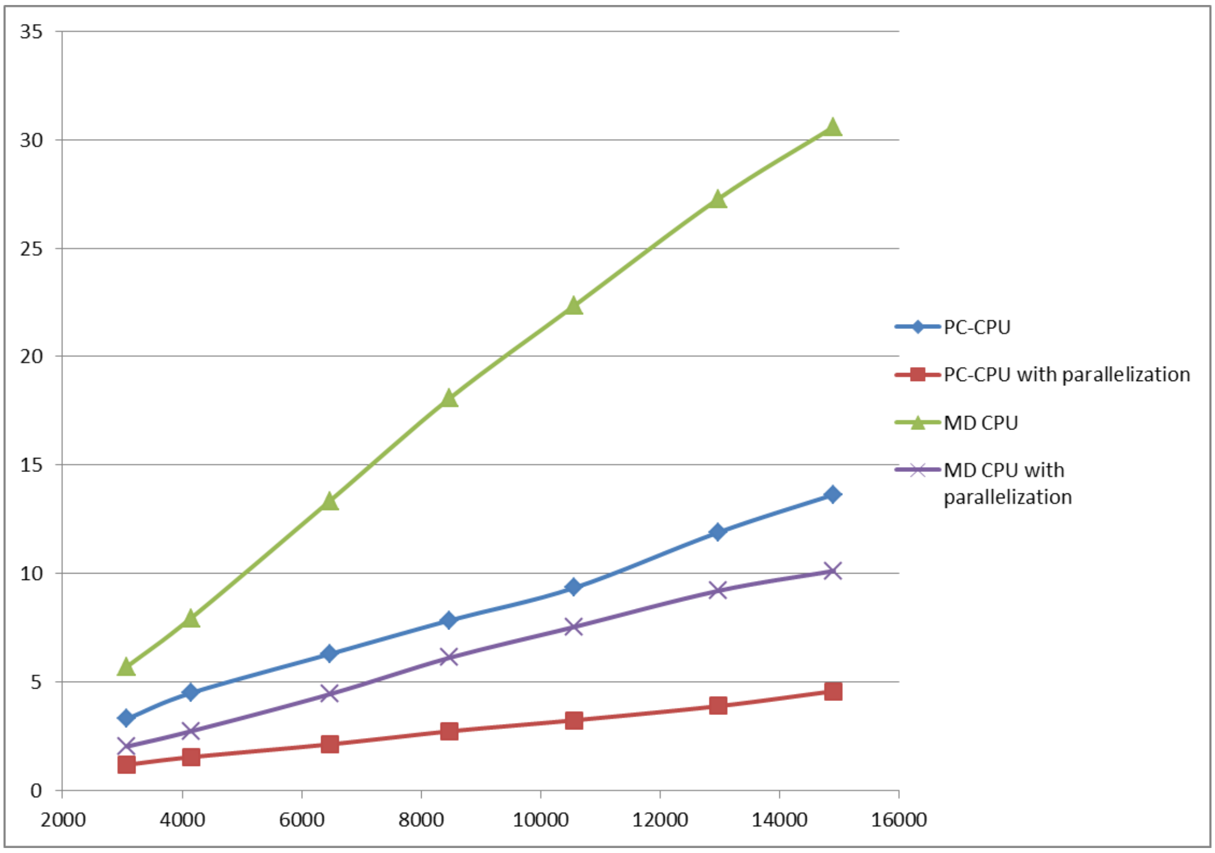

Figure 6: Time per iteration vs number of cloth vertices

CONCLUSIONS AND FUTURE WORK

This paper presented a technique for efficient garment simulation on a mobile device. It implements a mass-spring system with velocity modification to overcome super elasticity and image space based approach exploiting the GPU for collision detection. The following more important conclusions can be drawn:

- Today's mobile devices possess sufficient computational power to be used for virtual try-on garment simulation on a static human body.
- On devices with GPU supporting only OpenGL ES 2.0 the simulation can be accelerated by parallelizing the computations on the multi-core CPU, using the SIMD instructions and utilizing the GPU for collision detection.

The experiments showed that dressing a virtual body with a pair of jeans with a reasonable resolution takes two seconds on a modern smart phone.

REFERENCES

[1] Dinh H.T., Lee C., Niyato D., Wang P. (2012). A Survey of Mobile Cloud Computing: architecture, Applications, and Approaches. *Wireless Communications and Mobile Computing* 13 (18): 1587–1611.

[2] Wang Y., Chen I.R., Wang D.C. (2015). A Survey of Mobile Cloud Computing Applications: Perspectives and Challenges. *Wireless Personal Communications* 80 (4): 1607-1623.

[3] Terzopoulos D., Platt J., Barr A., Fleischer K. (1987). Elastically deformable models. ACM Proceedings of SIGGRAPH 21 (4): 205–214.

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

[5] Vassilev T.I., Spanlang B., Chrysanthou Y. (2001). Fast cloth animation on walking avatars. *Computer Graphics Forum* 20 (3): 260–267.

[6] Press W.H., Teukolsky S.A., Vetterling W.T., Flannery BP. (1992). Numerical Recipes in C, 2nd. edition. Cambridge University Press.

[7] Baraff D. & Witkin A. (1998). Large steps in cloth simulation. *Computer Graphics Proceedings, Annual Conference Series*, 43–54.

[8] Provot X. (1997). Collision and self-collision detection handling in cloth model dedicated to design garments. *Proceedings of Graphics Interface*, 177–189.

[9] Volino P., Magnenat Thalmann N. (1995). Collision and self-collision detection: Efficient and robust solutions for highly deformable surfaces. *Computer Animation and Simulation* 1995, Terzopoulos D, Thalmann D, (Eds.), Springer-Verlag, 55–65.

[10] Baciu G., Wong W. S., Sun H. (1999). Recode: an image-based collision detection algorithm. *The Journal of Visualization and Computer Animation* 10 (4): 181–192.

[11] Myszkowski K., Okunev O.G., Kunii T.L. (1995). Fast collision detection between complex solids using rasterizing graphics hardware. *The Visual Computer* 11 (9): 497–512.

[12] Vassilev T.I., Spanlang B. (2012). Fast GPU Garment Simulation and Collision Detection, *20th International Conference on Computer Graphics, Visualization and Computer Vision WSCG 2012*, Plzen, 19-26.

[13] Jeon J.H., Min S.D., Kong M. (2015). Implementation of Cloth Simulation Using Parallel Computing on Mobile Device. *International Journal of Electrical and Computer Engineering* 5 (3): 562-568.