Simulating Real-Time Cloth with Adaptive Edge-based Meshes

T. J. R. Simnett R. G. Laycock A. M. Day
School of Computing Sciences, University of East Anglia
Norwich, NR4 7TJ, UK
{t.simnett|robert.laycock|amd}@uea.ac.uk

ABSTRACT

We present an approach to simulating detailed cloth in real-time using an adaptive edge-based mesh, by enhancing its usability and performance. We show how simple seaming can be used to combine multiple adaptive meshes into garments, accomplished with the use of discontinuous material co-ordinates. Performance is improved by decoupling the mesh adaption and simulation steps, allowing efficient data structures to be exploited for the simulation and collision detection. Greater memory efficiency is achieved by pre-allocating a pool of memory to be used by any mesh at any hierarchical level. The collision detection process is integrated into the edge-based adaption technique, enabling a garment to be coarsened and refined repeatedly, such that no new vertices are created inside of the object in collision. Our technique is illustrated for a 67k triangle character wearing a T-shirt adaptively refined to 6k triangles in real-time.

Keywords: Adaptive mesh, cloth simulation, garment

1 INTRODUCTION

The real-time simulation of cloth and particularly clothing is a challenging task, both the deformation calculations and collision detection are expensive thus limiting what is achievable with finite resources. Of course very detailed and realistic cloth animations can be simulated, albeit offline with very fine meshes. However, they are limited by how long the user is prepared to wait for the result. We would like to be able to simulate detailed clothing in real-time, since many applications require it such as virtual reality, games and garment prototyping. Mass-spring networks are an approach made popular for simulating real-time cloth by Provot, where an inverse dynamic procedure to correct the lengths of super-elongated springs particularly enhanced its useabilty [Pro95]. Mass-spring networks are favoured for their speed and real-time applications compared to more complex physical models [DB99]; however, the cost of the simulation in either case is proportional to the size of the mesh. Taking into account hardware advances over the last ten years, more is achievable in real-time now but the quality of offline simulations will always exceed that of real-time simulations. An approach to narrow the gap between offline and real-time systems concerns the use of adaptive meshes. Instead of using a fine regular mesh, a much coarser mesh is used, which is subsequently adaptively refined only in regions considered most important. Important regions are those that require a greater density of polygons needed to model important aspects of visually realistic cloth; most commonly wrinkles in areas of high curvature, and to accurately resolve collisions with objects. Coarsening, being the opposite of refinement takes place in regions that no longer require a high density of polygons.

We continue to build on Simnett et al's [SLD09] work, improving the edge-based adaptive refinement approach, which works on triangular meshes specially enhanced with connectivity information. The subdivision is the result of splitting edges in the mesh, this is performed using application specific edge-based criteria. This approach allows very fast incremental updates to the mesh, whereby only two adjacent triangles require updating on each edge split to ensure a conforming mesh. We use a mass-spring network for our simulation, with Verlet numerical integration and, importantly, make use of Provot's inverse dynamic procedure to correct the lengths of super-elongated springs [Pro95]. The mass-spring network shares the same topology with the adaptive mesh, where the mass of each vertex is calculated as one third of the adjacent triangle's mass, which is updated as the mesh refines.

The main contributions of this paper are:

- Incorporating multiple configurations into the transition states of edge-based adaptive meshes for improved real-time performance.
- Multiple edge-based adaptive meshes are combined to form garments. A T-shirt, constructed from two meshes attached at their seams, is illustrated in real-time.
- Collision aware mesh adaption technique enables a garment to be coarsened and refined resulting in no vertices of the cloth penetrating the character's body.

Firstly we will survey existing work in related areas and the rest of the paper is organised into the following sections detailing our work: Edge-Based Adaptive Mesh and Configurations, Seaming, Adaptive mesh Memory, Adaptive Cloth and Collision Detection.

2 PREVIOUS WORK

Adaptive meshes are a good way of reducing the cost of simulating cloth without sacrificing heavily on the detail and have been used a number of times previously, with the adaption criterion most commonly based on curvature [VL03][LV05]. Villard and Boroucharki found their adaptive mesh improved simulation times as much as six times [VB05]. Memory usage and access can be particularly problematic in the case of large meshes where disk storage is required [VL03]. Seaming, that is the joining of meshes together, is of particular interest to cloth simulations, allowing actual garments to be constructed in a similar way to reality by way of physically stitching pieces of flat material together to form garments. Ma et al [MHB06] presented a method that automatically constructed a seam surface along an arbitrary path on the surface of an irregular mesh. They simulated seam puckering on 3D garments with a mass-spring network, which produced realistic wrinkles but it unfortunately requires the use of very fine meshes beyond the capability of real-time simulation. Pabst et al [PKST08] looked at the influence of seams on the bending of fabric, together with an accurate bending model. If using a relatively coarse mesh, they found that it must be locally refined around the seams to smooth out the abrupt changes that would otherwise be present in the bending stiffness between adjacent elements, thus increasing the costs. Their method produced excellent realism, comparing a real garment with that of a simulated one with each step taking 65ms to calculate the bending forces. Durapinar et al's [DG07] virtual garment design and simulation system featured automatic pattern generation by cutting a regular mesh along lines between defined corner vertices. A mass-spring system was used with seaming taking 4.313 seconds for a skirt with 1400 vertices, finalized by combining vertices into one by adding each vertex's spring forces to the other. Seaming is not always mentioned for character garments and no specific details were included for the garment using the adaptive mesh in [LV05], suggesting that a 3D mesh in the shape of the garment is used instead. We believe a simple approach is prudent for real-time simulation. We detail our approach in Section 4 that will allow seaming of an edge-based adaptive mesh inorder to allow garments to be created. Collision detection research is a large field, many algorithms and data structures have been designed for this. Without suitable collision detection and response, cloth simulations are severely limited and complex algorithms are used for high fidelity offline animations. Bridson et al.'s [BFA05] work generates high quality wrinkles with complex collisions. By robustly processing collisions, contacts and friction, with a collision aware post processing step a surface is subdivided and iteratively smoothed for rendering. Their approach took approximately two minutes per frame for a piece of cloth with 150 x 150 nodes. Self collisions for cloth is particularly expensive to compute. Govindaraju et al. [GKJ+05] presented a method that accurately detected all self-collisions for a 23K triangle cloth dress in 400-550 ms. Their chromatic decomposition partitions a fixed topology mesh into independent sets, and uses a linear-time culling algorithm performing 1D overlap tests on the CPU and a 2.5D on the GPU. Achieving real-time performance puts tight limits on the complexity of the cloth and collision testing. Fuhrmann et al. [FGL03] achieved interactive animation of cloth with self-collision by approximation; it considered only pairs of particles and held them apart using a bounding hierarchy of particles.

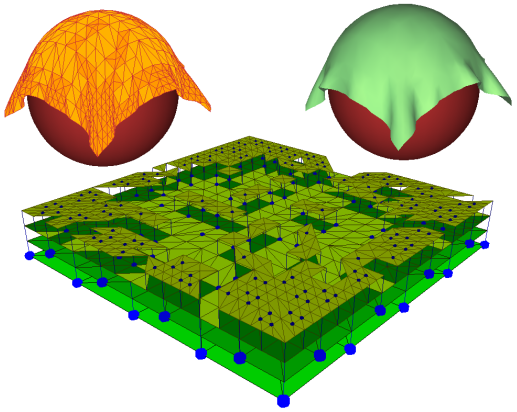

Figure 1: Cloth draped over a sphere, rendered with smooth lighting and coloured levels together with an illustration of the adaptive mesh's hierarchy.

3 EDGE-BASED ADAPTIVE MESH

The Edge-Based approach to adaptive meshes allows fast incremental changes to the mesh, refining or coarsening by one level at a time. Edges in the mesh are

split in order to subdivide the mesh, and curvature, edge length and edge collisions have proved suitable criteria for controlling this for cloth modelling. Coarsening of the mesh is achieved by rejoining previously split edges. When an edge is split, a new vertex is generated at the centre and two child edges are created. Triangles are dealt with using a state based retriangulation scheme, where the triangles adjacent to the split (or rejoined) edges are internally retriangulated. This is undertaken to build a conforming mesh with new additional internal edges and internal triangles. Upon reaching a full subdivided triangle (1-to-4 split), further refinement of child edges and triangles are allowed. The adaptive mesh forms a complex hierarchical tree structure of levels, with edges containing two child edges and a central vertex, triangles contain upto four internal triangles and three edges. The highest level in use of any part of the mesh constitutes the whole mesh used for the cloth simulation. Figure 1 shows a simple simulation of a piece of cloth draped over a sphere, for which the hierarchy is illustrated.

3.1 Configurations

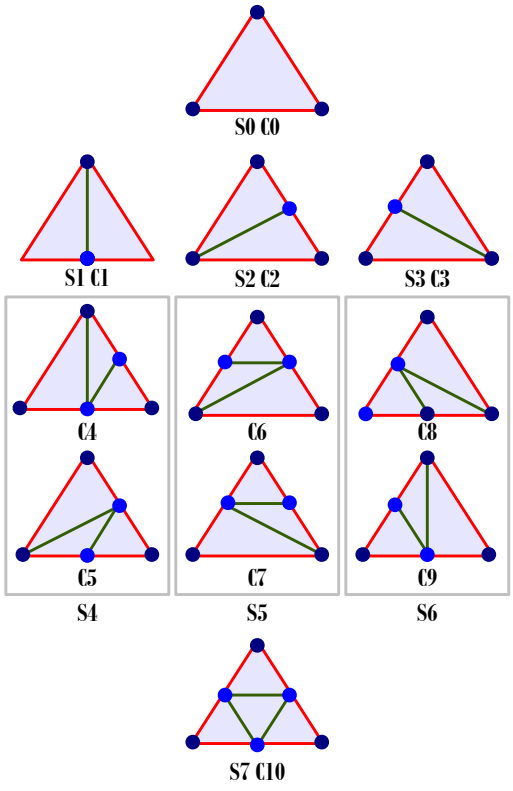

Figure 2: Triangle states and configurations showing internal triangulations of a parent triangle.

The state based triangulation approach enabled the fast determination of the new triangulation by checking the change in status of external edges [SLD09]. We have since expanded this approach to include the other three triangulations missing from the previously defined eight states. Each of the three new triangulations share the same state as an existing one, that is the status of the edges are the same. We therefore require an additional identifier other than state for the triangulations; we call this a triangle's "configuration". There are a total of eleven configurations starting with an empty parent triangle to a completely sub-divided one (1-to-4 split) and all the possibilities in-between. We map every state to a configuration, except in three cases where now each of those states maps to two possible configurations. See Figure 2 for the states and configurations.

Deciding on the configuration to use is straight forward in the case of the 1-1 mappings. For the other 3 states, we can consider a transition from the previous configuration to one that provides the minimal change, this helps to keep the forces more consistent between steps as the cloth adapts. For example: consider that when transitioning from C1 to C4/C5, the algorithm would choose C4 as the configuration that provides minimal change. Therefore, if C5 was chosen instead the change would be visually detectable in the rendered cloth (unless the vertices were co-planar).

It can be seen that many transitions between two configurations have common features. This is because it is not ideal to have to configure the whole parent triangle when it is re-triangulated. Instead, a lower level approach to re-triangulation that exploits the similarities is possible, although it increases code complexity considerably. The previous configuration is reused as much as possible, changing only as necessary. There are a total of 100 possible transitions that may be used by the adaptive mesh. Table 1 shows a comparison of the speed up that is achieved with this approach for a selected number of cases.

Transition	Time (μs)	Improved Time (μs)
C0 to C1	0.218079	0.199257
C1 to C4	0.316416	0.177083
C1 to C5	0.312889	0.236971
C2 to C5	0.312540	0.173625
C4 to C1	0.238927	0.070819
C4 to C10	0.435041	0.257016

Table 1: Time comparison of the new faster re-triangulation approach compared to the work in [SLD09].

Some transitions such as C0 to C4/C5, are both equally costly and we need to either use a default selection or select it based on some criteria. We have investigated the use of three simple approaches based on edge curvature, edge length and edge rest length. The two length criteria choose the configuration that gives the shorter new edge length, based on either

the deformed length or rest length. The curvature criterion looks to see which way the parent triangle is bending and selects the best configuration to conform to this. Figure 3 shows very small differences between different selection criteria. The reason for this is that in most cases there is a faster transition available that will be used in precedence to the selection criteria, such that it will have little effect overall. Therefore we consider the use of a default criteria, since it requires no additional processing while not negatively impacting on the refinement.

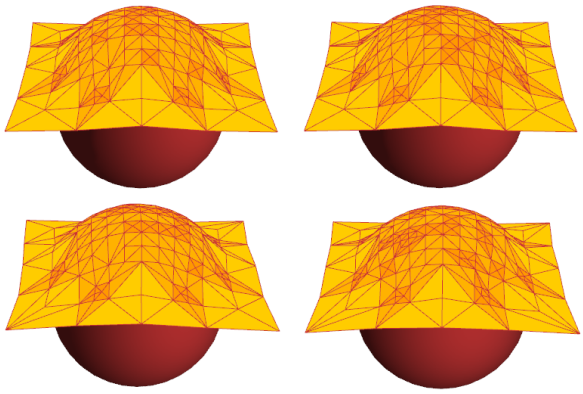

Figure 3: Four spheres are illustrated using four different transition selection methods. (Top Left: Default Choice, Top Right: Edge Length, Bottom Left: Edge Rest Length, Bottom Right: Edge Curvature) A selection method is used for transitions that are equally costly, otherwise the fastest minimal change transition is used.

4 SEAMING

Complicated models for seaming do not currently appear feasible for real-time simulations; therefore we take a simple approach and perform seaming on the base level of our adaptive mesh. We accomplish this by merging the vertices and connecting the edges of seams together. Firstly the user can interactively select links between groups of vertices and pairs of boundary edges using the mouse or alternatively use previously stored links. The mesh can be simulated using the coarse mesh if desired before actual seaming occurs to bring the meshes together, (linked seam vertices are constrained together). Next we perform the actual merging; where initially the vertices and edges are removed from the meshes. A new vertex is created for each group of vertices that are linked and is stored as part of the garment, each new vertex has a total mass of all the combined vertices together with a merged adjacent triangle list. The connectivity of the boundary edges and triangles in the meshes must be updated as to reference the new merged vertex rather than each of the group. Consequently, we can use the merged vertices instead of the old ones in the simulation. Finally boundary edges are connected into pairs, setting their pointers to point to each other, which are now stored in a seam edge as part of the garment. Seaming for a T-Shirt is illustrated in Figure 4, using a mesh for the front and one for the back with seams above the shoulders, under the arms and down the sides.

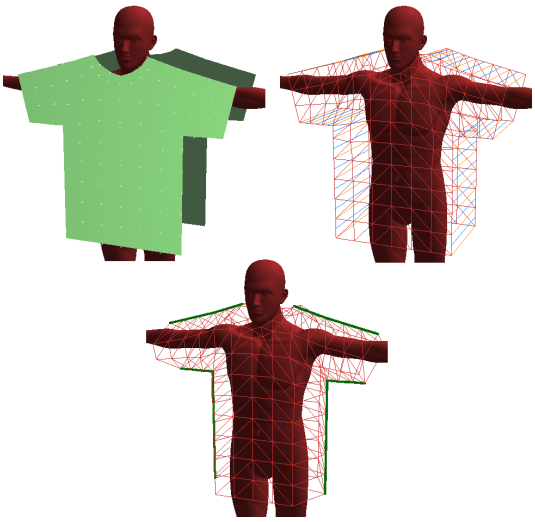

Figure 4: Interactive Seaming, Top Left: initial base meshes (rendered with normals), Top Right: seaming links defined (vertex links in orange, edge links in blue), Bottom Centre: after seaming, seams in green.

4.1 Discontinuous Material Co-ordinates

There is a challenge with managing material co-ordinates for the meshes; 2D material co-ordinates allow the quick determination of un-deformed lengths needed for the physics calculations between any two points of the mesh and they are readily acquired from flat textile patterns. Ultimately there exists discontinuous co-ordinates along the seam lines between multiple meshes and this is also the case for a single mesh (e.g. a single mesh seamed to form a cylinder). Storing the material co-ordinates in the vertices, alone cannot support this. We present a method that supports discontinuous co-ordinates between any base triangle in the mesh, and continuous co-ordinates are used within triangles as they are subdivided. To support this, material co-ordinates were needed to be moved from the vertices into the edges. Since an edge between adjacent triangles consists of two oppositely directed pieces, each side can have its own material co-ordinates. Where the material co-ordinates are continuous, both sides of the edge will hold references to the same material co-ordinates. In the case of discontinuous co-ordinates, each side will hold reference to a different set of co-ordinates. Figure 5 illustrates

this, three meshes are to be seamed together with three sets of continuous material co-ordinates. Vertices are merged but the material co-ordinates are not, then edges are connected together. It should be noted that the boundary edge lengths should match between pairs for best results, as one edge is designated as the control edge whose length will be used for the spring forces in the simulation.

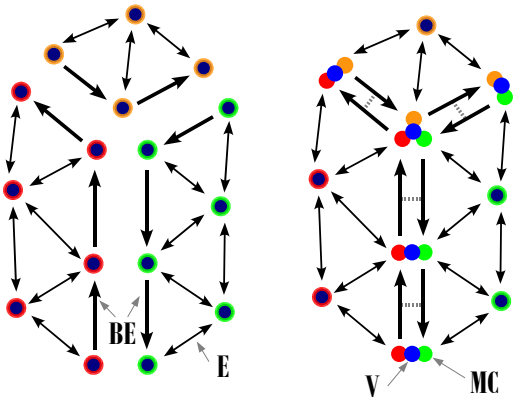

Figure 5: Three meshes each with their own set of material co-ordinates (red,orange,green), seamed together with discontinuous material co-ordinates (MC) along their boundary edges (BE). Vertices are shown in blue (V).

5 ADAPTIVE MESH MEMORY

Memory considerations are often overlooked in real-time simulations where we favour pre-computation wherever possible, and consider algorithm efficiency of most importance. However, adaptive meshes require a large amount of memory to store subsequent levels even for small base meshes. Real-time constraints often force the use of pre-allocation for all levels that may be needed, this is because the operating system's managed dynamic memory imposes a relatively large overhead for creation and deletion. In regards to real-time cloth, the purpose of the adaptive mesh is to allow refinement only in areas that require more triangles (typically high curvature). There is a limit on the overall number of triangles in a piece of cloth that can be simulated and rendered in real-time, and the adaption criteria should be chosen accordingly. We can make use of this to achieve large memory savings, by only pre-allocating the maximum memory needed by the mesh over the course of a real-time simulation. The memory is divided into pools, one for each of the principle building blocks of the mesh: triangles, edges, vertices and material co-ordinates. As the mesh is refined and coarsened the pools are used dynamically, in a similar way but with almost no overhead compared to system managed memory. We use a linked list approach with two lists, which allow constant time access to the pools. One list stores unused objects to be taken from the pool, and the other holds list nodes ready for when an object is put back onto the pool. There is an additional advantage to this approach, the pools may be shared simultaneously by multiple meshes; each mesh does not need significant resources allocated to it which may not be fully utilised. It is difficult to predict the exact requirements for a complex simulation especially with collisions, however, experimentation can yield good results. Taking the simulation in Figure 10 as an example, the base mesh requires 0.3 MB and an additional 7.2 MB in the memory pools was needed with allowances to support upto approximately 7,000 triangles. The full refinement to level 3 would require 15.6 MB in total, this means in this example, our approach has saved 8.3 MB or around 53% of the memory needed over that of pre-allocation for the adaptive mesh.

6 ADAPTIVE CLOTH

6.1 Separation of Adaption and Simulation

The original approach in [SLD09] was to perform both adaption and simulation in the same algorithm and do this at each update. We have found this not to be efficient in cases of very small time steps, often these very small steps are needed to ensure numerical stability of the simulation when using finer meshes. All the edges in the mesh will be checked against the criteria for each time step for adaption. When very few changes have taken place, the cost of checking will surpass the actual cost of updating the mesh structure per step. Therefore, we have since decoupled the adaption and simulation allowing more resources for the simulation to run at a rate required for stability. The adaptive mesh will update a constant number of times each second (typically at 30Hz), we refer to this as a major step. Minor steps refer to the simulation steps, which are performed after each major step, the number of these depend on the time step required to give stable results. Collision detection and response is performed once at the end of each major step rather than every minor step to save time when vertices are only moving very small distances.

To further increase performance, it is not necessary to recalculate the triangle normals (using the cross product) and vertex normals (average of adjacent triangles) during the adaption step since the vertices are not moved. Also, new vertices are initially co-linear with their parent edge, and therefore fast interpolation is sufficient to determine their normal.

6.2 Data Structure Traversal

Our adaptive mesh's hierarchy enables an efficient algorithm for refinement but imposes an overhead for the

simulation where the hierarchical data structure must be traversed many times each step. We have alleviated this cost by constructing a temporary list of pointers to triangles, edges and vertices each time the adaptive mesh is updated, they are particularly effective now that the adaption is decoupled and the lists may be valid and used for a number of simulation steps. Table. 2 shows the time in milliseconds for each simulation step, the cost of creating the list, and the improved simulation timing that results from using the list, hence the cost of the data structure traversal to the simulation can be seen.

Level	Simulation (ms)	List Creation (ms)	Sim. with List (ms)
Base	0.049832	0.001453	0.033908
1	0.214280	0.006593	0.136197
2	0.880447	0.033203	0.573544
3	2.964748	0.136163	2.407764

Table 2: Simulation times for a single step using standard traversal and temporary lists updated each adaptive step. A base mesh of 128 triangles is used and the cost of list creation can be seen.

7 COLLISION DETECTION

Although geometric shapes provide straight forward collision checks by use of their parametric equations, building complex objects out of them is not easy. The decision was made to implement a general method that can be used with any 3D triangle model. The only requirement is that the triangles of the mesh must be outward facing (determined by the vertex winding) forming a surface where the cloth will not be allowed to penetrate through into the object. A problem with the highly flexible nature of cloth is that even if the vertices are not penetrating the object, edges and triangles may intersect the object's surface causing very noticeable visual artefacts. The use of the adaptive mesh, greatly increases the cloths ability to approximate the underlying surface it is in contact with, but this alone is not sufficient. A full triangle-triangle collision approach provides a potential solution although at a cost to processing compared to only vertex-triangle collision detection. We seek a compromise for use with real-time simulations; a suitable approach is to leave a region above the surface to hide these intersections. If too great a region is used, the gap will be very noticeable and unrealistic; relatively too small and intersections will still be visible. To enable this, we use a separate mesh for collision and rendering. The original mesh is loaded, and then sent to the graphics card to be stored in a vertex buffer object. To construct the collision mesh, we first calculate smooth normals for the vertices (which may be the same as used for rendering). Using these normals we expand the mesh by moving the vertices in the direction of their normals with an adjustable offset. The result is a shell around the original object, giving us the region to hide intersections within. The originally loaded mesh can be deleted from main memory if desired, leaving it only on the graphics card for rendering so that memory requirements are not doubled by using the collision mesh. We allow the mesh to be arbitrarily rotated and translated, by calculating and storing the transformations using a 3x3 matrix for the rotation and a 3D vector for the translation. The cloth is simulated in world co-ordinates, and vertices are first transformed into the object's local coordinate space to undergo the collision checks. Considering a cloth vertex (mass) has a current position and a previous position (which was outside of the object), the collision with a static object can be determined by checking that the vertex's path has not intersected the object.

The object is potentially a large triangle soup, and therefore we initially must partition it into a more efficient structure for collision testing inorder for real-time computation. A 3D grid of cells is used, where each cell is an axis aligned bounding box that contains a list of all the triangles that are inside or partially overlapping.

The main collision algorithm between a cloth vertex and a object proceeds as follows. The previous and current position are transformed into the object's coordinate space, the transformed points are hence referred to as A and B. Each grid Cell that the path AB intersects is tested for collisions by testing the vertex with each triangle within the cell. Figure 6 illustrates the vertex triangle collision test, firstly the segment AB is intersected with the triangle's plane to find an intersection point (IP). If IP exists, the barycentric co-ordinates are calculated to determine if this lies within the triangle face; if this is so, the surface point (SP) is calculated by projecting B in the direction of the triangle's normal. The current position is moved to the SP after transforming it back into world co-ordinates.

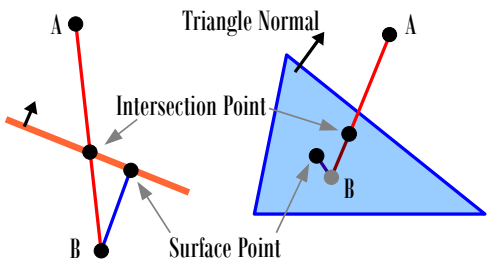

Figure 6: Vertex-triangle collision: The vertex's current position (B) is moved to the surface point, if the intersection point is within the triangle.

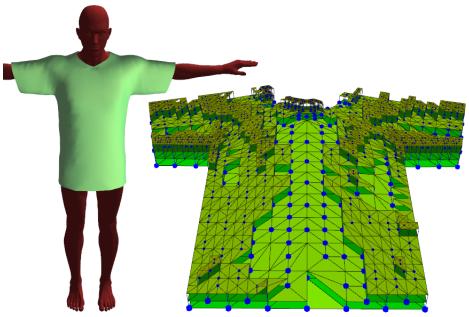

Figure 7: A T-Shirt draped on a static character, with the adaptive hierarchy of the front mesh illustrated.

7.1 Collision within Mesh Adaption

Collision routines are called from within the adaption step, as necessary, in a number of places. When an edge is split, the newly created vertex in the centre of the edge can be created inside other objects, a situation that is prevented under normal circumstances. We must be able to determine when this has occurred and find a suitable location to relocate the vertex to. We tackle both of these problems in one step, by constructing a ray pointing outwards from the object and testing it for collision with the collision mesh. The direction we use is the average of the edges two end normals. If there is a collision, then the vertex is moved to the intersection point adjusting the previous position to preserve the velocity, see Figure 8. Figure 9 shows three snap shots of the cloth simulation on a character, and demonstrates how the coarse base mesh may be used to speed up the initial placement of the garment.

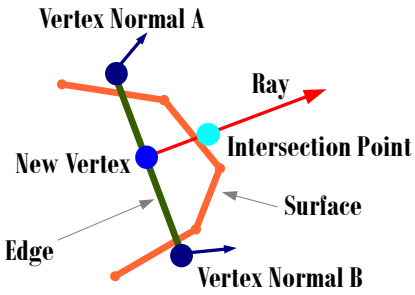

Figure 8: Resolving a new vertex that has been generated inside an object. It is moved to the intersection point between the directed ray and the surface.

7.2 Conflicting Criteria: Edge Collision and Curvature.

When using multiple criteria to control the refinement an coarsening of the mesh, problems can arise from these conflicting. An edge may be in collision and therefore should be split, but the curvature criteria may determine that the edge should not be split and would

Figure 9: Left: coarse mesh, Centre: immediately after two adaption steps (incremental approach implies a change of one level for each step and level two is required in some areas to resolve the collisions), Right: after 1 second (30 adaption and 120 simulation steps), the cloth has relaxed.

therefore be rejoined in the next step. Previously Simnett et al. [SLD09] used a rejoin wait count, with the aim to prevent rapid flipping between coarsening and refining an edge by specifying a number of steps before an edge was allowed to rejoin after having been split. This was although visually effective, caused the flipping to still occur each time the rejoin wait was completed. We have changed how it is used in order to improve upon this and the performance. If edge collision splitting is enabled for the cloth; before an edge is rejoined, it is checked for collisions. The purpose being is that if there is an edge collision and it is rejoined, it will almost certainly be split again at the next step. The edge in collision will be prevented from rejoining, saving the costs of the edge join and retriangulation. The additional collision check has a cost associated with it too, so we make use of the rejoin wait count again but now it saves the cost of collision and curvature checks. However, there is a trade off as rapid coarsening can bring force, integration and collision savings for the simulation. A balance between a too long rejoin wait and not long enough is important; we find a rejoin wait count of ten steps works well, which will basically spreads out the cost of the rejoin checks over the ten steps as they are not synchronized.

8 RESULTS

We implemented our cloth simulation using C++ and used Opengl for rendering, the results presented in this paper were performed using a PC with a Intel 2.66 GHz Core i7 920 using a single thread with 6GB of RAM. The limit of our approach is approximately seven thousand triangles, with the simulation times being the limiting factor where mesh adaption takes less than 10% of the total time for each major step. Figure. 10 shows a mesh of this size, worn as a T-shirt by a static character that is simulated in real-time. Each major step takes 22ms running at 30Hz, with 4 minor or simulation steps running at 120 Hz. Figure 7 shows the corresponding adaptive hierarchy for the front mesh of the T-Shirt. The accompanying video features the character, where two meshes are initially seamed together and the cloth is simulated in real-time. To demonstrate the adaptive mesh and collision detection more robustly, a

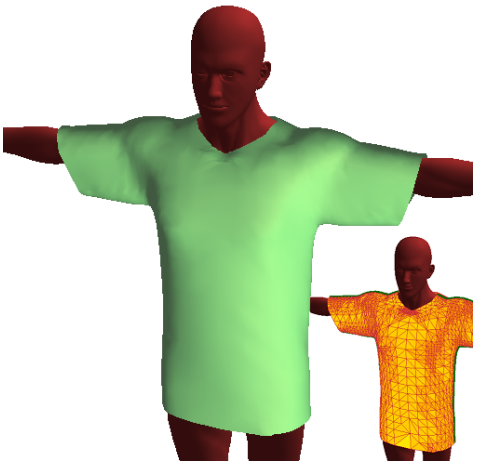

Figure 10: A T-shirt is draped on a static character with 67k triangles. The cloth is constructed from two base meshes seamed together, which totals 316 triangles, and are adaptively refined to 6199 out of a possible 20224 (Level 3) triangles. It takes approximately 22ms for each adaption and 4 simulation steps, running at 30Hz in real-time.

force on the cloth is introduced from a movable virtual fan. The video can be found at the following url: http://www.crowdsimulationgroup.co.uk/Cloth.wmv

9 CONCLUSION

We have presented a number of improvements to the edge-based adaptive mesh approach presented in [SLD09], such as increased performance by decoupling mesh adaption from simulation, using a more efficient data structure for traversal and introducing new configurations to speed up re-triangulation. Furthermore, we have implemented a robust collision detection scheme for the cloth with arbitrary triangle meshes and have integrated this into the adaption process, which together with seaming allows the real-time simulation of a detailed T-shirt on a virtual character. The improved handling of memory has provided a saving of 53% for this simulation, compared to the pre-allocation of the complete hierarchy. In the future we would like to simulate garments on moving characters, and also on multiple characters simultaneously. We anticipate changes will be needed to our collision handling in order to allow non-static objects, and the grid based approach will need to be modified to support jointed characters. Our work shows that the adaptive mesh would work well with additional level of detail approaches, since an off-screen character can be simulated using purely their base mesh. Figure 9 shows that our refinement method could quickly return to detailed garments if the character were to come back into view. The fact that the base meshes are small in terms of memory requirements compared to the refined mesh, allows the memory pool to be used by multiple meshes effectively.

REFERENCES

[BFA05] R. Bridson, R. Fedkiw, and J. Anderson. Robust treatment of collisions, contact and friction for cloth animation. In *International Conference on Computer Graphics and Interactive Techniques*. ACM Press New York, NY, USA, 2005.

[DB99] M. Desbrun and A. Barr. Interactive animation of structured deformable objects. In *In Graphics Interface*, 1999.

[DG07] F. Durupinar and U. Gudukbay. A virtual garment design and simulation system. In *IV'07. 11th International Conference*, pages 862–870, 2007.

[FGL03] A. Fuhrmann, C. Gross, and V. Luckas. Interactive animation of cloth including self collision detection. *Journal of WSCG*, 11(1):141–148, 2003.

[GKJ+05] N.K. Govindaraju, D. Knott, N. Jain, I. Kabul, R. Tamstorf, R. Gayle, M.C. Lin, and D. Manocha. Interactive collision detection between deformable models using chromatic decomposition. *Proceedings of ACM SIGGRAPH 2005*, 24(3):991–999, 2005.

[LV05] L. Li and V. Volkov. Cloth animation with adaptively refined meshes. In *Proceedings of the Twenty-eighth Australasian conference on Computer Science-Volume 38*, pages 107–113. Australian Computer Society, Inc. Darlinghurst, Australia, 2005.

[MHB06] Liang Ma, Jinlian Hu, and George Baciu. Generating seams and wrinkles for virtual clothing. In *VRCIA '06: Proceedings of the 2006 ACM international conference on Virtual reality continuum and its applications*, pages 205–211, NY, USA, 2006. ACM.

[PKST08] Simon Pabst, Sybille Krzywinski, Andrea Schenk, and Bernhard Thomaszewski. Seams and bending in cloth simulation. *VRIPHYS*, 382(1):24–41, 2008.

[Pro95] X. Provot. Deformation Constraints in a Mass-Spring Model to Describe Rigid Cloth Behaviour. In *Graphics Interface*, pages 147–147. Canadian Information Processing Society, 1995.

[SLD09] T.J.R Simnett, S.D. Laycock, and A.M. Day. An edge-based approach to adaptively refining a mesh for cloth deformation. In *EG UK Theory and Practice of Computer Graphics 2009*, pages 77–84, 2009.

[VB05] J. Villard and H. Borouchaki. Adaptive meshing for cloth animation. *Engineering with Computers*, 20(4):333–341, 2005.

[VL03] V. Volkov and L. Li. Real-time refinement and simplification of adaptive triangular meshes. *VIS 2003. IEEE*, pages 155–162, 2003.