

Detail Preservation for Simulated Characters

1 Summary

Flesh simulations for animated characters are often run at much lower resolution than would be required to reproduce all the details of the animation, especially in regions such as the face and hands. These details are often critical to conveying the actions and emotions of the character, so it is important to be able to add these details back to the simulation after it is run. Doing this robustly requires a geometric analysis of the conversion process from input animation to the simulation mesh, and a method for reapplying the missing detail to possibly very different simulation results.

2 Background

We are given an input animation consisting of an animated surface mesh. Parts of the mesh represent important character details that have been hand animated, such as facial expressions, the motion of fingers, etc. Other parts of the mesh may be bulk features of the character, for example the jiggly belly of a large character, or the floppy ears of a running dog. These are the types of features we wish to animate using physically-based simulation, and the goal is to do so while maintaining the carefully hand animated details. One approach to this problem was applied for the human characters in *WALL•E* [2]. However, that approach suffered limitations hindering its usefulness, and would perform poorly on general characters where the secondary motion is proximate to the detailed animation. We would like to investigate some alternative approaches to the problem that should yield better results.

3 General Procedure

The software we will be using for the purposes of the project is PhysBAM. Many of the tasks detailed below are already supported in PhysBAM, but certain steps in the procedure will require implementation. Given an input animated surface mesh, the general procedure we will use is as follows:

1. Generate an appropriate simulation mesh.
2. Compute target positions for the surface of the simulation mesh.

3. Compute target positions for the interior of the simulation mesh.
4. Generate constraint patches in the simulation mesh to be used during simulation.
5. Run the simulation.
6. Map the simulation results to the animation mesh to get the final result.

More details on these steps are given below.

3.1 Generating a Simulation Mesh

The input animation mesh is a surface mesh only, so a volumetric simulation mesh needs to be created for the simulation. We will use existing tools in PhysBAM to create a simulation mesh. In practice, the simulation mesh will consist of approximately 10K-40K tetrahedra, with a target frame rate of 1-2 s/frame running on four processors. As far as developing the method, the critical feature of our simulation mesh is that it *not* resolve all the details - it should be coarser than the input surface mesh.

3.2 Determining the Target Simulation Surface

In this step, we will map the source animation to the surface of the coarse simulation mesh. The surface of the simulation mesh will not exactly match the input animation surface, so we will need some way of transferring the animation from the input surface to the simulation surface. The resulting positions are the target animation for the surface of our simulation mesh (corresponding to the target input animation mesh), to which we will add secondary motion. One question is then, how should we compute the target animation for the surface of the simulation mesh? One possible approach to doing this is to use the deformation transfer method of [5]. In this approach, surface deformations of a source triangle mesh are mapped to surface deformations of a target triangle mesh. A correspondence must be established between triangles of the two meshes. The correspondence is symmetric, so we could also use this deformation transfer to map the final results from the simulation mesh back to the animation mesh in Step 6.

A key desirable feature of any chosen method is that the mapping $source \rightarrow target \rightarrow source$ be an identity. This will generally not be the case for the approach of [5], so we will have to correct for any error in the mapping. For example, we can compute the error in the above mapping and use it as a correction in Step 6.

3.3 Determining the Target Simulation Interior

Once we have surface target positions from Step 2, we will generate volume target positions. These interior targets will help to guide the interior of the

mesh during simulation for better correspondence with the input animation. We can use a quasistatic simulation to find the interior target state. The quasistatic simulation will use the surface target as a boundary condition for a steady state solve. This will give us the right rest pose for the simulation, because if the character was allowed to settle around that pose, it would match the target surface state exactly. Furthermore, the quasistatic simulation is quite fast and this process can easily be run independently in parallel for all the frames. We can use the quasistatic solver in PhysBAM, which is described in [6].

3.4 Defining Constraint Patches

We will use the method described in [1] to constrain the simulation to match the targets determined in Step 2 and Step 3. We do not want to constrain all of the degrees of freedom in the simulation, but only some so that it tracks the target animation while still producing interesting dynamic effects. The method in [1] describes a way of doing this. Various patches are created on the simulation mesh, and the simulation state will match the target state in an *averaged* sense over these patches. Finer patches can be created where we want the simulation to match the target more closely. Coarser patches will be used where the simulation should have more freedom to determine the motion. We will want to define patches in the interior of the simulation mesh as well as patches including the surface. We can think of the interior patches as acting like a type of skeleton constraining the motion on a coarse level.

3.5 Running the Simulation

We will use the Newmark time integration scheme in PhysBAM to run the simulation.

3.6 Mapping Simulation Results to the Input Animation

Once we have run the simulation, constrained to match the targets, we wish to map the results back to the input animation mesh to produce the final result. A desirable property is that if there is no deformation in the target pose determined in Step 2, the mapping back will not alter the animation surface. This is a restatement of the above identity property mentioned. We will pursue the possibility of applying this reverse mapping using the deformation transfer technique as described above, with an error correction. A possible approach to applying this error correction is to use the approach in [4] or [3] for local shape preservation. This would involve transforming the different geometric poses of our animation mesh into a coordinate space that facilitates blending or combining different poses. Once in this space, we can compute the error caused by mapping from the input animation mesh to the coarse simulation surface and back, add this correction to the simulation result, and transform the combined result back into geometry.

References

- [1] Miklós Bergou, Saurabh Mathur, Max Wardetzky, and Eitan Grinspun. Tracks: toward directable thin shells. *ACM Trans. Graph.*, 26(3):50, 2007.
- [2] G. Irving, R. Kautzman, G. Cameron, and J. Chong. Simulating the deformed: Finite elements on walle. In *SIGGRAPH 2008 Sketches & Applications*. ACM Press, 2008.
- [3] Yaron Lipman, Daniel Cohen-Or, Ran Gal, and David Levin. Volume and shape preservation via moving frame manipulation. *ACM Trans. Graph.*, 26(1):5, 2007.
- [4] Yaron Lipman, Olga Sorkine, David Levin, and Daniel Cohen-Or. Linear rotation-invariant coordinates for meshes. *ACM Trans. Graph.*, 24(3):479–487, 2005.
- [5] R. Sumner and J. Popović. Deformation transfer for triangle meshes. In *ACM Trans. on Graph. (Proc. ACM SIGGRAPH)*, volume 23, pages 399 – 405, 2004.
- [6] J. Teran, E. Sifakis, G. Irving, and R. Fedkiw. Robust quasistatic finite elements and flesh simulation. *Proc. of the 2005 ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.*, pages 181–190, 2005.