Efficient Collision Detection for Example-Based Deformable Bodies

Ben Jones
University of Utah
benjones@cs.utah.edu

Joshua Levine
University of Arizona

Tamar Shinar
University of California, Riverside

Adam W. Bargteil
University of Maryland, Baltimore County

ABSTRACT

We introduce a new collision proxy for example-based deformable bodies. Specifically, we approximate the deforming geometry as a union of spheres. During pre-computation we perform a sphere packing on the input, undeformed geometry. Then, for each example pose, we move and resize the spheres to approximate the example. During runtime we blend together these positions and radii, using the same skinning weights we use for the geometry. We demonstrate the method on a car crash example, where we achieve an overall speedup of 5-20 times, depending on the resolution of the collision proxy geometry.

CCS CONCEPTS

• **Computing methodologies → Simulation by animation**; **Physical simulation**;

KEYWORDS

Example-based Simulation, Collision Detection, Skinning, Local Blending

ACM Reference format:
Ben Jones, Joshua Levine, Tamar Shinar, and Adam W. Bargteil. 2017. Efficient Collision Detection for Example-Based Deformable Bodies. In *Proceedings of MiG '17, Barcelona, Spain, November 8–10, 2017,* 5 pages.
https://doi.org/10.1145/3136457.3136469

1 INTRODUCTION

Robust detection and handling of collisions is a vital component of simulation systems. While most render geometry is represented using triangle meshes, that representation poses significant challenges for collision detection and response. While techniques and software exist that work directly with triangle meshes, they often require very small timesteps, are computationally expensive, and must avoid pitfalls arising from finite-precision arithmetic. As a result, interactive simulators often use collision proxy shapes that approximate the shape of the render mesh and are more stable and numerically efficient. These proxy shapes are commonly constructed as a union of convex shapes, such as spheres or discrete oriented polytopes (K-DOPs). Unfortunately, computing such a decomposition can be computationally expensive, so this approach is insufficient for simulating objects that can undergo arbitrary deformations.

In this work, we describe a method for computing and maintaining such a convex decomposition for example-based deformable bodies. At run time these objects deform into a blend of artist-provided deformations. Since the example deformations are known a priori, we can precompute convex decompositions for each deformation and efficiently compute the deformed convex decomposition as a blend of the example proxies.

2 RELATED WORK

A comprehensive review of collision handling is beyond the scope of this paper, but we recommend the survey by Teschner and colleagues [2004]. Because collision detection with convex geometry is far simpler and faster than geometry with concavities, static objects are typically broken up into a convex decomposition—a set of convex pieces that are equivalent to the original geometry. Unfortunately, this approach breaks down with deforming geometry not only because the intial decomposition no longer represents the deformed geometry, but also because deformation can introduce new concavities, so simply updating the collision geometry requires more complex and expensive collision detection algorithms. One approach to collision detection of deformable bodies in the context of real-time simulation is to develop highly optimized algorithms [Civit-Flores and Susín 2015]. Another approach is to adopt the classic technique of creating simple collision proxies for complex geometry [Baraff and Witkin 1997; Hubbard 1996]. Typically these proxies are made up of many convex pieces. Unfortunately, when geometry deforms these proxies are no longer valid and must be updated. Like work with model reduced elastic bodies [Barbic and James 2010; Kim and James 2011] and the work of Spillmann and colleagues [Spillmann et al. 2007], who assume a deformation model based on shape matching, we take advantage of the fact that our deformations are not entirely general. In our application, runtime deformations are determined by a blend of example deformations, enabling us to derive a simple and efficient approach for deforming collision proxies.

3 METHOD

Our method decomposes an arbitrary simulation object, whose surface is represented as a triangle mesh, into a set of overlapping

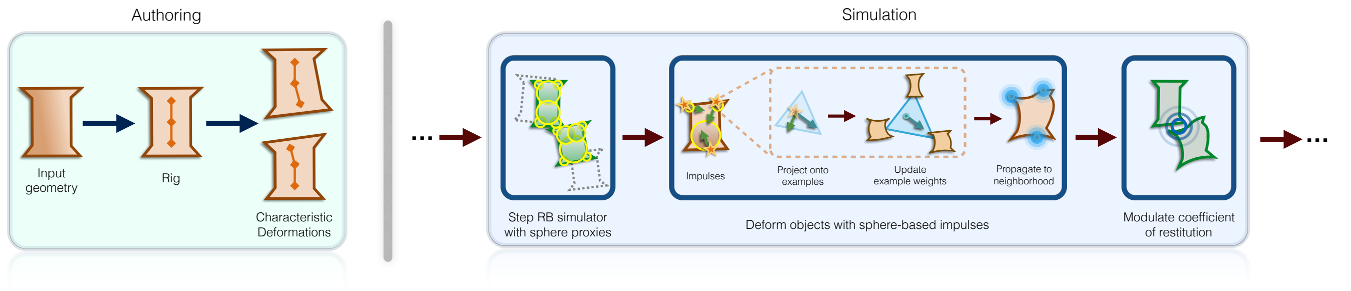
Authoring
Input geometry
Rig
Characteristic Deformations
Simulation
Step RB simulator with sphere proxies
Impulses
Project onto examples
Update example weights
Propagate to neighborhood
Deform objects with sphere-based impulses
Modulate coefficient of restitution

Figure 1: *Overview of the authoring and simulation process, depicting the example-based deformation system of Jones and colleagues combined with our novel collision proxy geometry (yellow).*

spheres. As the object deforms at runtime, the spheres are moved and resized so that the surface of their union approximates the deformed geometry of the object. To implement this efficiently, our algorithm consists of a precomputation step and an efficient run time update.

3.1 Input and Notation

As input, our method requires:

- an undeformed triangle mesh, $\mathbf{M}_0$
- a set of e deformed meshes, $\mathbf{M}_1, \ldots, \mathbf{M}_e$, the example deformations of the object
- a function, f that maps an interior vertex x_0 from inside $\mathbf{M}_0$ to its deformed position in each $\mathbf{M}_i$.

In our implementation, all meshes share the same topology. We tetrahedralize $\mathbf{M}_0$ before deforming it into meshes $\mathbf{M}_1, \ldots, \mathbf{M}_e$, and f is computed by barycentric interpolation. However, this implementation choice is not required by our method.

3.2 Simulator Dynamics

To motivate the remainder of our method, we will briefly describe the simulation framework we used in our implementation.

We implemented our collision detection and response system within the example-based plastic deformation simulator of Jones and colleagues [2016]. For completeness, we provide a brief overview of the simulation method but refer the reader to [Jones et al. 2016] for further details.

Deformed meshes, $\mathbf{M_i}$ are computed by rigging the undeformed mesh, $\mathbf{M_0}$, with control handles and using linear blend skinning. To ease authoring, we tetrahedralize the mesh and compute skinning weights via the method of Jacobson and colleagues [2011]. Since the interior tetrahedral mesh vertices are skinned as well, we can compute our map, f, by using barycentric interpolation on the tetrahedral vertices.

The simulator is based on the Bullet rigid body simulator. The motion of objects is modeled as completely rigid (i.e., the degrees of freedom are a single translation and rotation per object). Plastic deformation is incorporated by modifying the shape of the object based on the collision impulses generated during simulation. We make use of Bullet's *compound* collision shape which uses a bounding volume tree to accelerate collision queries.

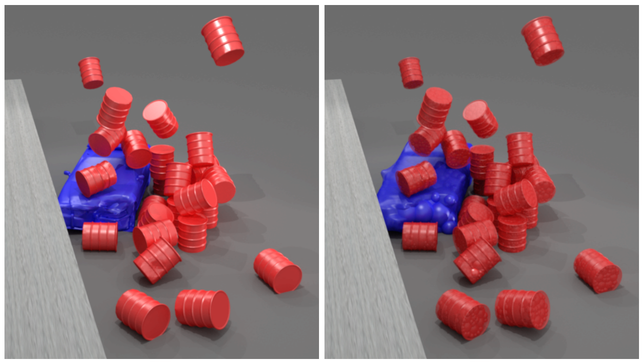

Figure 2: *Car crash scene with render geometry (left) and collision proxies overlaid on render geometry (right)*

At each tetrahedral mesh vertex, we compute the skinned position via linear blend skinning. However, the handle transforms vary per-vertex. The handle transform at a given point is computed by blending the handle transforms of the example poses.

While our prototype is built on a simulator that incorporates only plastic deformation, our approach is applicable to elastic objects as well. Also, our method is applicable to any deformation model where a few representative or extreme deformations are known in advance and can be locally interpolated.

3.3 Precomputation

Our precomputation step outputs a set of spheres, S_0, as well as e deformations of that set, $\mathrm{S}_i, i \in [1..e]$. Each deformation contains the same number of spheres, with modified positions and radii. For each sphere set, S_i, the surface of their union approximate the corresponding mesh, $\mathbf{M}_i$.

3.3.1 Initial Collision Proxy (S_0). We begin by computing S_0 from $\mathbf{M}_0$ by adapting the approach of Budsberg and colleagues [2014]. We compute a signed distance field of $\mathbf{M}_0$ and iteratively add spheres to the set S_0. Each sphere we add is centered at the minimum of the signed distance field, following the convention that

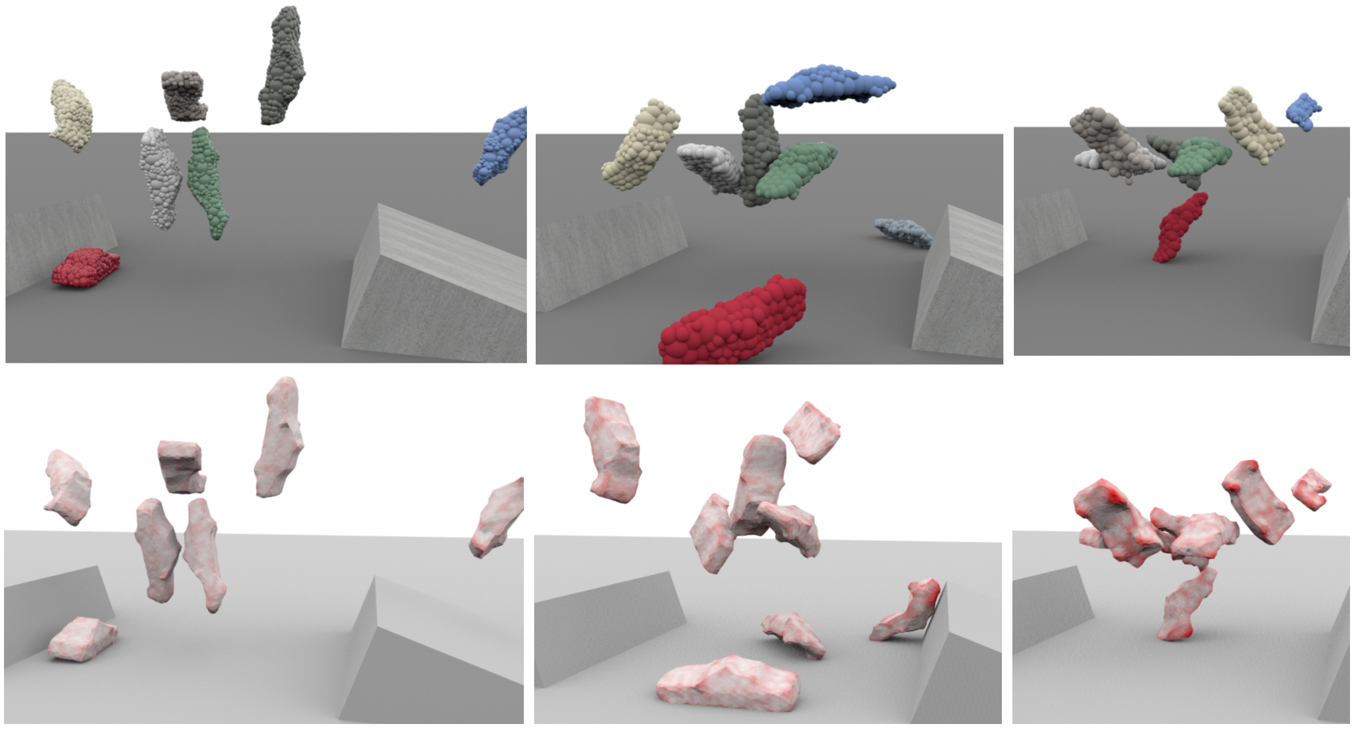

Figure 3: *Cars crash into each other after launching off ramps. From left to right, 320, 114, and 66 spheres per car. Decreasing the number of spheres improves performance but increases error, visualized in red, below.*

interior points are negative and hence the center is maximally far from the surface. We set the radius of this sphere to be the minimum of a user-defined maximum radius and the original distance from the center to the surface. The signed distance field is then updated by setting the values at all points inside the sphere to be 0. This prevents our algorithm from creating a new sphere whose center lies within an existing sphere, resulting in a proxy with fewer spheres.

This procedure terminates when the next sphere to be added would have a radius smaller than a user-defined minimum size or the total number of spheres exceeds a user-defined maximum. Note that since sphere radii are determined by the distance to the triangle mesh, rather than the distance to existing spheres, the spheres will overlap and be entirely contained with the surface.

The number and size of spheres is controlled by modifying the resolution of the signed distance field as well as the three user parameters for total number of spheres and the minimum and maximum allowed sphere radii. These controls allow the user to trade off between the accuracy of the collision proxy and the computational cost of collision detection. For the examples in this paper, we set the maximum number of spheres to be infinity, so the the minimum sphere radius is the only active termination criteria. In a resource constrained application, the maximum number of spheres may be a more convenient parameter.

3.3.2 Example Collision Proxies (S_i). Once S_0 is computed, the deformed sphere sets S_i are computed. For each sphere in S_0, the corresponding sphere center in S_i is computed via the map, f. Its radius is computed via the same procedure as above: the minimum of the user defined maximum radius and the distance to $\mathbf{M}_i$.

As centers move, it becomes possible that the sets S_i could contain gaps. This situation is mitigated by the fact that the sphere overlaps are typically deep in S_0. In our examples, gaps that did occur did not affect simulation quality(errors are quantified in Figures 5 and 6). Gaps could also be reduced by using a smaller maximum radius value, at the cost of requiring additional spheres.

3.4 Runtime Collision Proxies

As the object deforms, we update the position and radii of the spheres by blending the precomputed deformations S_i. We assume that the current deformation of the object is a barycentric blend of the undeformed pose and the e example deformations. We compute the current sphere positions and radii as a barycentric blend of $\mathrm{S}_i, i \in [0..e]$.

These blends need not be global. In our implementation, each tetrahedral mesh vertex stores the barycentric coordinates of the blend at that point. To deform our spheres, we use the barycentric coordinates of the nearest tetrahedral mesh vertex in the initial pose.

4 RESULTS AND DISCUSSION

Table1 gives timing information for the example in Figure 3, where a number of cars are launched at each other. Notably, even with the highest resolution collision proxies, our approach results in a 5× speedup; with a very coarse but still passable approximation,

Table 1: *Timing in ms for 10 seconds of animation for the car crash example. For comparison, the final row gives timing information for the method of Jones and colleagues [2016]. Even at the highest resolution our collision proxies reduce computation time by a factor of five.*

# Spheres	Rigid Body Sim	Total
320	29.4	37.3
114	10.4	13.7
66	6.6	10.5
Triangle Mesh	151.9	210.0

we achieve a 20× speedup relative to triangle based collision detection with approximately 1500 triangles per car. Figure 5 plots the median and maximum error during the simulation, computed as the distance from input mesh vertices to the surface of the sphere proxy surface. Because the dynamics and deformations are driven by collisions, the resulting animations differ slightly. Figure Figure 2 shows an additional example where a car crashes into a pyramid of barrels.

In Figure 4, we show a set barrels being dropped onto a loading dock. The number of spheres decreases from left to right by increasing the minimum allowed sphere radius. With fewer spheres, the collision proxies do not fill in the sharp corners of the barrel, but still produce reasonable results. The median and maximum errors are plotted in Figure 6.

We note that because our approach requires a water-tight input mesh to create the distance field, our proxies are not based on the high-resolution render geometry, but rather a coarser, water-tight approximation. As a result, some spheres can be seen poking through the render geometry of the car.

4.1 Future Work

While our method improves performance relative to the baseline of performing triangle-mesh collisions, there is room for further optimization. In particular, it may be possible to accelerate collision queries using a custom bounding volume structure rather than the more general implementation in Bullet.

While the size of gaps between spheres is easily controllable for the undeformed mesh, $\mathbf{M}_0$, the sphere approximation is less accurate during deformation, as shown in Figures 5 and 6. To improve the surface approximation in the deformed meshes, it may be possible to construct a signed distance for the deformed meshes and spheres. Then, by following the method from Section 3.3.1, we can compute spheres that fill in gaps in the deformed shape, which can then be mapped back to the rest space via the inverse map, f^{-1}. This would increase the accuracy of the approximation at the cost of additional precomputation, and the use of more spheres.

ACKNOWLEDGMENTS

This work was partially supported by the National Science Foundation under Grant No. IIS-1314896, IIS-1654221, and IIS-1314813

REFERENCES

Baraff, D. and Witkin, A. (Eds.). 1997. *Physically Based Modeling: Principles and Practice.* ACM SIGGRAPH 1997 Course Notes, Vol. 19. ACM SIGGRAPH.

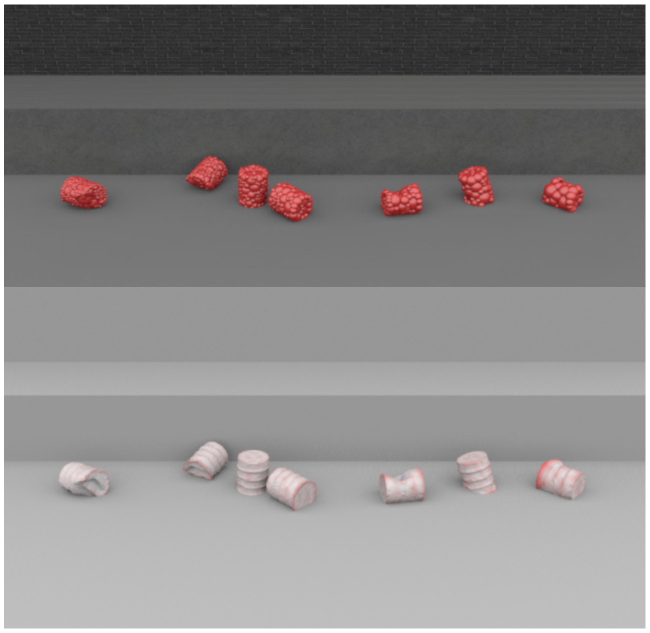

Figure 4: *Barrels with a decreasing number of spheres dropped on a loading dock. The top row shows the error measured as the distance from the mesh to the sphere proxy surface. The barrel with the fewest spheres has the highest error near the sharp edges of the barrel.*

Barbic, J. and James, D. L. 2010. Subspace self-collision culling. *ACM Trans. Graph.* 29, 4, 81:1–81:9. https://doi.org/10.1145/1833351.1778818

Budsberg, J., Zafar, N. B., and Aldén, M. 2014. Elastic and Plastic Deformations with Rigid Body Dynamics. In *ACM SIGGRAPH Talks.* Article 52, 1 pages. https://doi.org/10.1145/2614106.2614132

Civit-Flores, O. and Susín, A. 2015. Fast contact determination for intersecting deformable solids. In *Proceedings of the 8th ACM SIGGRAPH Conference on Motion in Games, MIG 2015, Paris, France, November 16-18, 2015.* 205–214.

Hubbard, P. M. 1996. Approximating polyhedra with spheres for time-critical collision detection. *ACM Transactions on Graphics (TOG)* 15, 3, 179–210.

Jacobson, A., Baran, I., Popović, J., and Sorkine, O. 2011. Bounded Biharmonic Weights for Real-time Deformation. *ACM Trans. Graph.* 30, 4, 78:1–78:8.

Jones, B., Thuerey, N., Shinar, T., and Bargteil, A. W. 2016. Example-based plastic deformation of rigid bodies. *ACM Trans. Graph.* 35, 4, 34.

Kim, T. and James, D. L. 2011. Physics-based Character Skinning using Multi-Domain Subspace Deformations. In *Proceedings of the 2011 Eurographics/ACM SIGGRAPH Symposium on Computer Animation, SCA 2011, Vancouver, BC, Canada, 2011.* 63–72.

Spillmann, J., Becker, M., and Teschner, M. 2007. Efficient updates of bounding sphere hierarchies for geometrically deformable models. *Journal of Visual Communication and Image Representation* 18, 2, 101–108.

Teschner, M., Kimmerle, S., Zachmann, G., Heidelberger, B., Raghupathi, L., Fuhrmann, A., Cani, M.-P., Faure, F., Magnenat-Thalmann, N., and Strasser, W. 2004. Collision Detection for Deformable Objects. In *Eurographics 2004, State-of-the-Art Report.* Eurographics Association, 119–135.

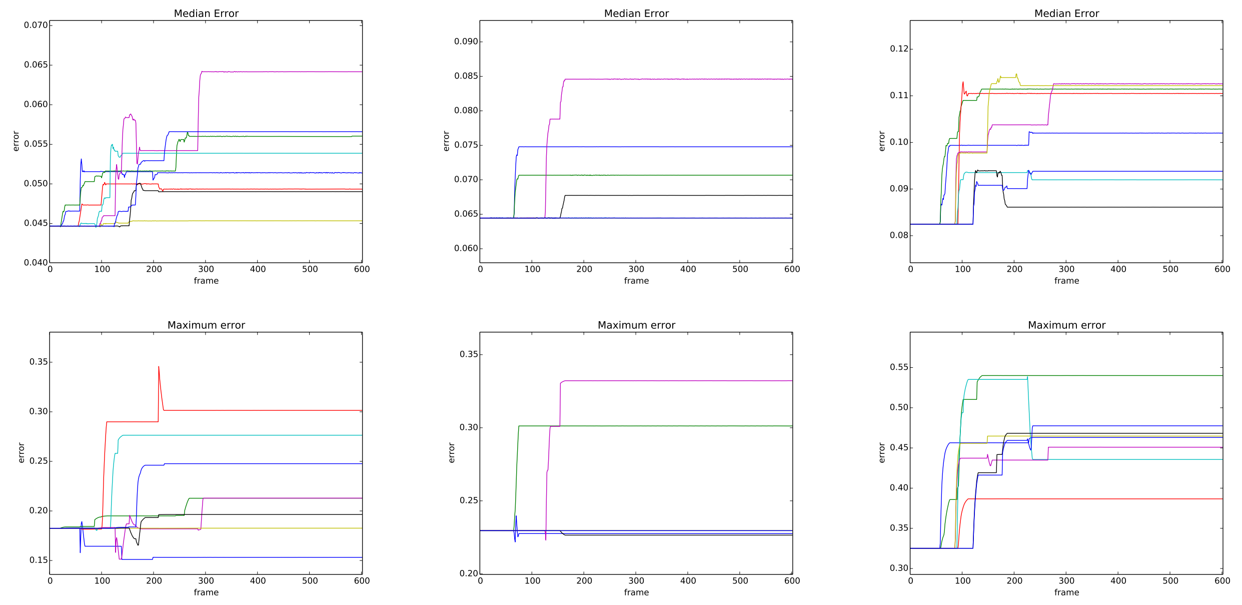

Figure 5: *Median error (top row), and maximum error (bottom row) for the animation from Figure 3 with (left to right) 320, 114, and 66 spheres. Error measured as distance in meters from the triangle mesh vertices to the sphere proxy surface. The car is 5.25 meters long. Each color represents one of the 8 cars in the scene.*

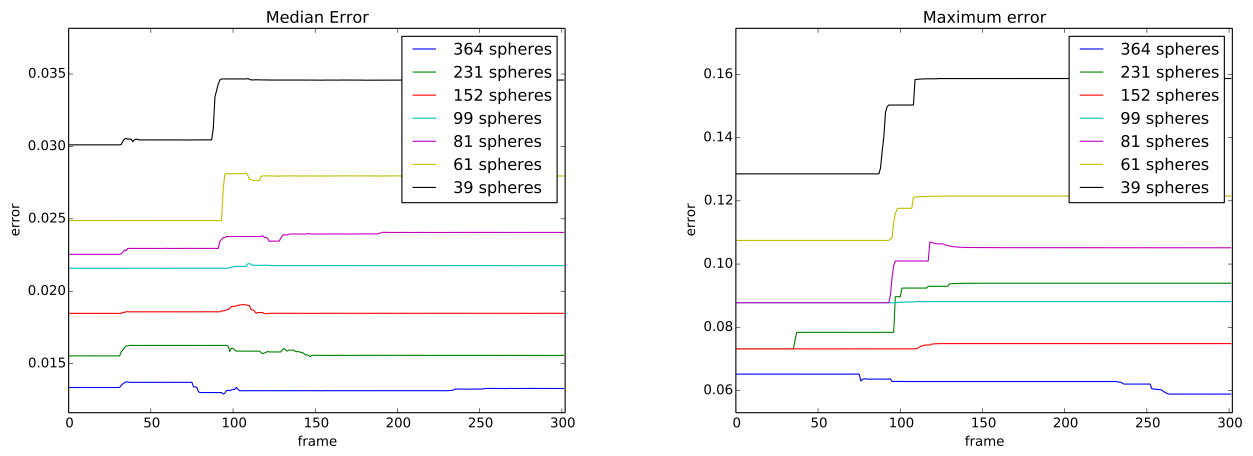

Figure 6: *Median error (left), and maximum error (right) for the animation from Figure 4. Error measured as distance in meters from the triangle mesh vertices to the sphere proxy surface. The barrel is 0.72 meters tall.*