This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2018.2859924, IEEE Transactions on Visualization and Computer Graphics

Real-Time Collision Detection for Deformable Characters with Radial Fields

Sebastian Friston, *Member, IEEE,* and Anthony Steed, *Member, IEEE*

Abstract—Many techniques facilitate real-time collision detection against complex models. These typically work by pre-computing information about the spatial distribution of geometry into a form that can be quickly queried. When models deform though, expensive pre-computations are impractical. We present radial fields: a variant of distance fields parameterised in cylindrical space, rather than Cartesian space. This 2D parameterisation significantly reduces the memory and computation requirements of the field, while introducing minimal overhead in collision detection tests. The interior of the mesh is defined implicitly for the entire domain. Importantly, it maps well to the hardware rasteriser of the GPU. Radial fields are much more application-specific than traditional distance fields. For these applications - such as collision detection with articulated characters - however, the benefits are substantial.

Index Terms—distance fields, collision detection, deformable meshes, articulated characters

1 Introduction

Collision detection is fundamental to virtual worlds - to their physical plausibility and the logical rules by which they operate [1]. Collision detection is used in both online and offline applications. However, online applications have the additional constraint that collision detection and response must be computed in a limited time.

Real-time collision detection techniques compute information about the spatial distribution of geometry and store it in a form that can be quickly queried. For example, a tree structure that partitions traditional primitives, or an alternative representation of the geometry itself. These structures facilitate excellent performance, but the pre-computation stage can make them impractical when models begin to deform. Some techniques have been modified to support deformable models. This is typically done by finding ways to reduce the computation time required to update the structures, but real-time performance is still a challenge.

One popular structure is the distance field. A distance field represents a surface by storing the shortest distances to it in a regular grid. Distance fields are memory intensive, and very expensive to compute naïvely, even offline. Once computed however, the distance to a surface can be immediately queried for a point almost instantly, making them valuable for real-time collision detection. Techniques have been derived to accelerate the computation and update of distance fields, and a number of these achieve interactive rates [2], [3], [4]. However, interactive may still not be fast enough for applications such as games, that can dedicate only a fraction of the frame budget to collision detection.

In this paper, we propose a variation of distance fields with an alternative parameterisation. *Radial fields* store distances to a surface, but in 2D cylindrical space, rather than 3D Cartesian space. The parameterisation makes assumptions about the geometry, making radial fields more application specific than distance fields. However it significantly reduces their memory and computation requirements. Delineation of the mesh interior is automatic, and the parameterisation maps well to the hardware rasteriser of a GPU.

Radial fields are only suitable for models that can be decomposed into relatively well-tessellated star-convex elements. For suitable models however, such as articulated characters, the performance gains are substantial. We demonstrate the practicality of our system by implementing it in a real game engine - Unity 5.4 - and integrating it with a position-based dynamics cloth simulation. We compare the performance of radial fields to techniques representing the state-of-the-art in deformable object collision detection across a set of simulations of varying complexity. We show that radial fields outperform traditional distance fields by more than an order of magnitude, and outperform Spatial Hashing and Bounding Volume Hierarchy accelerated triangle-based tests by a factor of two.

- *Sebastian Friston is with University College London. E-mail: sebastian.friston.12@ucl.ac.uk*
- *Anthony Steed is with University College London. E-mail: a.steed@ucl.ac.uk*

Manuscript received December 15, 2017; revised May 16, 2018.

2 Previous Works

Collision techniques can be separated into two categories: some operate by clustering primitives to reduce the number of intersection tests, while others replace the primitives with data structures on which collision detection can be performed directly.

2.1 Collision Detection for Deformable Models

Many high-performance collision detection techniques have been developed, but most are designed for rigid bodies and include pre-computations that preclude deformable objects. In this section we review the subset of techniques formulated for deformable models. Teschner et al. [5] provide a survey of such techniques. We use their classification and review the latest developments in each area. They

This work is licensed under a Creative Commons Attribution 3.0 License. For more information, see http://creativecommons.org/licenses/by/3.0/.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2018.2859924, IEEE Transactions on Visualization and Computer Graphics

identify five approaches that fall generally into the two categories above. Bounding Volumes, Spatial Subdivision & Stochastic clustering based approaches reduce the number of intersection tests, while Distance Fields and Image Space approaches provide alternative representations.

2.1.1 Bounding Volumes

A good overview of Bounding Volume and Spatial Partitioning techniques is provided by Ericson [1] and Zachmann & Langetepe [6]. Bounding Volume techniques partition primitives into a set of volumes such as boxes or spheres that support quick intersection tests. This accelerates the collision detection process by enabling quick culling of large groups of non-colliding pairs, minimising the number of expensive per-primitive tests. Bounding Volume Hierarchies (BVHs) place these volumes into tree structures to accelerate the process further. The volumes used (e.g. Bounding Boxes or Spheres) vary as there is a trade-off between fit and the time taken to compute the volume.

As objects deform BVHs must be updated. Algorithms can be optimised if the temporal behaviour of the deformable object is predictable. Larsson & Akenine-Möller's [7] Dynamic Bounding Volume Hierarchy took advantage of temporal coherence by dynamically resizing existing Axis Aligned Bounding Boxes (AABBs). In some cases BVHs can be constructed for animations ahead of time [5], [8]. Other techniques rebuild the entire tree in real-time. Wald [8] presented a Surface Area Heuristic (SAH) BVH algorithm for use in ray-tracing animated scenes.

For articulated characters, further assumptions can be made. Mujika et al. [9] used trees of dynamically sized spheres, assigned using skinning weights. Redon et al. [10] used swept spheres with trees of Oriented Bounding Boxes. Easier support for continuous collision detection and self-intersection are two advantages of BVH approaches over many alternative representations.

Some authors have mapped construction and refinement to GPUs. Karras & Aila [11] restructured existing BVHs on GPUs by refactoring isolated *tree-lets* in parallel. Lauterbach et al. [12] used space filling (Morton) codes to quickly sort and cluster primitives on the GPU. He et al. [13] constructed BVHs on the GPU for models undergoing topological changes (e.g. during crash simulations). Meister & Bittner [14] applied *k*-means clustering on the GPU to build BVHs. This involves iteratively clustering and merging primitives.

While these offer significant improvement over equivalent CPU based approaches, most of them are designed for ray-tracing applications, and none claim to support real-time interaction. Lauterbach et al. [15] extended their hybrid space filling technique, supporting tighter Oriented Bounding Boxes for collision and distance queries in surgery simulations. Tang et al. [16] reformulate the problem as one of stream compaction (removing non-colliding elements from a set), which maps well to GPUs and achieves interactive rates for inter and intra-object collision detection for deformable models.

2.1.2 Spatial Subdivision

Spatial partitioning is similar to bounding volumes, but partitions space itself, rather than primitives. Space is divided using schemes such as Octree or Binary Space Partitioning trees, hashing, or simple grids. Again there is a trade-off, this time between traversal complexity, memory and cell size.

Teschner et al. [17] used spatial hashing functions for real-time collision detection. They achieved performance approaching that of uniform grids but with a reduced memory footprint. An advantage of uniform subdivisions for deformable objects is that they are independent of the underlying geometric complexity [5]. If the distribution of geometry is uneven enough however, non-uniform subdivisions can be more efficient. Wong et al. [18] presented a GPU implementation for constructing an adaptive octree grid that achieved interactive rates. Taking advantage of the limited deformation of articulated models, Rumman et al. [19] presented a technique to perform differential updates of a uniform subdivision using spatial hashing.

2.1.3 Stochastic Methods

Stochastic methods augment existing techniques by eliminating tests that have a low probability of success. They are based on the observation that to the human eye, plausible collision responses are qualitatively indistinguishable from exact. These approaches can never support exact or physically correct collision detection without effectively disabling them [5]. While many techniques are not exact, non-deterministic accuracy is a problem for applications such as cloth simulation, as penetration artefacts are highly salient and frame-to-frame differences in collision response can result in oscillations.

2.1.4 Image Space

Image space techniques project geometry to accelerate either the broad-phase or narrow-phase collision detection stages. Teschner at el. [5] list many examples that project geometry into image space to form depth maps. These maps can be used for interference testing. Use cases for both stages have undergone continued development. Faure et al. [20] projected layers of complex models in three orthogonal axes to determine overlaps. Jang et al. [21] used the GPU to quickly cull potentially intersecting pairs of triangles in the overlapping regions of AABBs computed by the CPU.

Rodriguez-Navarro et al. [22] used the depth maps to directly resolve collisions between cloth particles and an articulated character. Multiple viewpoints were placed around the body to produce depth maps against which particles were directly tested. This is similar to Vassilev et al. [23], who used whole body depth maps. The avatars were animated, but the pose was such that body parts did not occlude each other.

Image space techniques provide some independence from underlying object complexity. There is also the potential for improved performance utilising the rasterisation hardware of a GPU. What is most interesting for deformable objects however is that these techniques do not require a pre-processing stage. Our technique is similar to Rodriguez-Navarro et al.'s. However we use field space rather than local-image space to gain the advantages of distance fields and avoid the depth-complexity artefacts present in image space techniques.

This work is licensed under a Creative Commons Attribution 3.0 License. For more information, see http://creativecommons.org/licenses/by/3.0/.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2018.2859924, IEEE Transactions on Visualization and Computer Graphics

2.1.5 *Distance Fields*

A popular volumetric data structure is the Distance Field. Distance fields store the shortest distances to a surface in a Cartesian grid [6]. Originally used for rendering [24], [25], Jones et al. [26] describe many applications of distance fields from morphology of erosion, to visualisation, and boolean modelling operations. The advantages of distance fields over traditional boundary representations is that they define both the interior and exterior of the space in which the object sits. This allows the distance of a point to an object to be rapidly queried, independent of the complexity of the original geometry [27].

As a variant of voxelisation, distance fields are memory intensive, especially compared to other volumetric approaches such as sphere packing (e.g. [28], [29]). As a corollary, computing the field is both bandwidth and compute intensive. The naïve way - determining the minimum distance between every cell and primitive in turn - is prohibitively expensive even as a pre-processing stage, therefore a number of techniques have been developed to accelerate it. Examples include hierarchical techniques on the original geometry, various transforms that calculate the distance of voxels/cells from their neighbours, or propagate distances like wavefronts. Jones et al.'s survey [26] provides a comprehensive history of distance fields and a discussion of field computation techniques for rigid bodies.

Distance fields also have significant advantages however. They are independent of the original geometry and allow the collision response to be easily computed along with detection. Importantly queries are not only fast, but deterministic. Fuhrmann et al. [30] noted how this could be advantageous to collision detection, and applied distance fields for this purpose in particle-based physics simulations.

These advantages have seen authors persist in finding ways to support deformable models. Fisher & Lin [4] presented a technique for partial-updates of distance fields using bounding boxes around local deformations. Updating distance fields is essentially a 3D rasterisation. Accordingly a number of techniques have taken advantage of the GPU. Pantaleoni [31] introduced VoxelPipe - a GPU accelerated triangle voxelisation pipeline. Gascon et al. [32] used tetrahedral mesh rasterisation to update voxel grids during deformations of volumetric objects. ElBadrawy et al. [33] introduce a novel alternative called *inclusion fields* that store whether a cell is inside or outside the mesh, rather than the absolute distance. This allows the field to be updated quickly with a 3D rasterisation, but requires more work to compute a collision response. McAdams et al. [34] use the field as an acceleration structure: the nearest surface point is found in undeformed space, then projected into deformed space. This avoids re-computation, but introduces potential error proportional to deformation.

2.2 Radial parameterisations

Volumetric structures are traditionally memory intensive. Detailed geometry requires high resolution grids for accurate discretization. Adaptive resolutions [2] can improve memory efficiency, but make dynamic updates even harder. Like BVH techniques that take advantage of an object's topological or skeletal structure, alternative parameterisations can reduce memory usage [35]. Matching the shape of an embedded coordinate system to that of the object being represented uses the parameterisation itself to store geometric information. Like us, a number of authors have done this using radial or spherical parameterisations.

Fünfzig et al. [36] used a hierarchy of bounding boxes in spherical space to partition general models. Wong et al. [37] performed radial view tests of primitive clusters from observer points on a skeleton. These were designed to filter potentially colliding primitives in a CPU broadphase test however, whereas our technique is an entirely alternative geometric representation.

Carr et al. [38] represented geometry by fitting signed-distance functions to imperfect 3D scanner data for interpolation and extrapolation, while Koschier et al. [35] fit polynomials piecewise to spatially subdivided Signed Distance Fields. These representations are highly efficient and have the nice property of being differentiable anywhere. Fitting the functions however is non-trivial making the approaches unsuitable for deformable models.

Moustakas et al. [39] used analytical surfaces (in this case superquadrics) combined with sampled distances to reduce the effects of fitting error. The technique was extended by Vogiannou et al. [40] with multi-layered depth maps, and spheres in place of superquadrics. Theirs is most similar of all to ours. Sampled distances make these approaches more amenable to quick updates than purely analytical representations. These works however did not explore dynamic updates, GPU implementations, or compare their performance with existing collision detection techniques.

Our exact parameterisation itself may not be new. Kurihara et al. [41] alluded to a cylindrical distance field representing the head and shoulders of a human for quick collision queries in hair simulation. Kurihara et al. however did not provide any implementation details or performance comparisons. In this work we explore how a representation based on this parameterisation can be GPU accelerated in order to support deformable models, and how its performance compares with existing collision detection techniques.

3 Overview

We propose that radial fields (distance fields in cylindrical space) can provide the benefits of radial parameterisations, while also being amenable to fast updates on the GPU, making them suitable for collision detection with deformable models.

A radial field is defined by an axis in 3D space, divided into a set of uniform cells across its circumference and length. The field is computed by transforming faces of a polygon-soup mesh into 2D cylindrical space around the axis and rasterising them as if rendering to a traditional frame buffer. Each cell in the space is 'rendered' with the distance between the face's plane and the axis. An intersection is detected by identifying the cell containing a point, and comparing the distance of this point to the axis with the stored distance. If smaller than the distance, the point is inside the mesh, otherwise it is outside.

Objects must be generally cylindrically shaped, or there will be discontinuities in field resolution across the mesh.

This work is licensed under a Creative Commons Attribution 3.0 License. For more information, see http://creativecommons.org/licenses/by/3.0/.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2018.2859924, IEEE Transactions on Visualization and Computer Graphics

There is also limited support for non star-convex geometry. For many assets these limitations will be prohibitive. For particular assets though, such as humans, animals and some machines, they are inconsequential while the benefits could justify an application-specific data structure.

In our expected use-case, a set of radial fields are applied coincident with the bones of an articulated avatar. The fields overlap, creating a closed approximation of the surface. The fields are updated every frame, or when the surface is expected to have deformed. Together the fields support fast particle collision detection as shown in Sections 6 & 10.

4 Radial Fields

We first describe how radial fields are defined and how to transform between the pertinent coordinate systems in Section 4. We then describe how the rasterisation pipeline is re-purposed in Section 5. Finally we describe how collision detection is performed and how a collision response may be computed in Section 6.

4.1 Field Definition

A field is defined by an axis (w) in 3D space between two points A and B (Figure 1b). This axis should follow the principal 3D component of the mesh to ensure an even distribution of samples across the surface. Where a mesh is closed around the end (e.g. the tips of digits or the head) the axis should pass just beyond this. As will be seen, polygons perpendicular to the axis are not supported but are handled gracefully so they do not result in artefacts. In addition to w, orthogonal normal (u) and tangent (v) vectors are defined. These vectors synchronise the phase of the polar coordinate with the Cartesian system.

We define two coordinate systems for radial fields, *field space* and *cylindrical space*. Field space refers to the Cartesian coordinate system defined by (u, v, w) above. Imagine the coordinate system such that the z-axis lies along the field vector (w), the y-axis along the normal vector (u) and the x-axis along the tangent (v) (Figure 1c). World-space coordinates can be converted to this space with a typical transform matrix. From this space they can be converted into cylindrical space (Equation 1b). Cylindrical space is defined by two normalised parameters: z, the offset along the axis, and θ the angle around the axis, from the normal vector (Figure 1d) (Equation 2).

4.2 Coordinate Transformation

To convert from world-space to cylindrical-space, coordinates are transformed so they align with the basis vectors defining the cylindrical space (Figure 1a to Figure 1c). (u, v, w) form the basis vectors for this space, and so a change of basis matrix (B') can be defined the typical way. To convert to field space from object space, first the offset A is subtracted and the resulting points transformed by the change of basis matrix.

Fields may be attached to a moving object (Figure 1a). This must be taken into account when rasterising world-space geometry (Section 4.3) or detecting collisions with world-space particles by first removing its transform (Equation 1a). The final transforms then, to and from world-space, are given by Equation 1b and Equation 1a, respectively.

$$T_{toFieldSpace} = B' \cdot T_{Translate}(-A) \cdot {T_{object}}^{-1} \tag{1a}$$

$$T_{toWorldSpace} = T_{object} \cdot T_{Translate}(A) \cdot B \tag{1b}$$

Once the points are in field space, they can be converted to and from cylindrical space (Equation 2 and Equation 3).

$$\begin{aligned} d &= \sqrt{x^2 + y^2} \\ \theta &= \frac{1}{2\pi} \begin{cases} \arctan_2(y, x) + 2\pi, & \text{if } \arctan_2(y, x) \leq 0 \\ \arctan_2(y, x), & \text{otherwise} \end{cases} \\ z &= z \end{aligned} \tag{2}$$

$$\begin{aligned} x &= d\cos(\theta) \\ y &= d\sin(\theta) \\ z &= z \end{aligned} \tag{3}$$

4.3 Articulated Meshes

For articulated meshes, fields are defined coincident with an avatar's bones. That is, T_{object} in Figure 1 is the bone's world transform. On each frame the mesh is baked into world-space. When the field space transform (Equation 1a) is calculated during rasterisation it includes the inverse of the bone transform, removing any deformations due to the bone itself and leaving only those due to adjacent bones. This approach is independent of the actual skinning method, so long as the application maintains an animated skeleton. Further, it implicitly supports deformations from any other sources, so long as they can be baked into the mesh.

5 Implementation

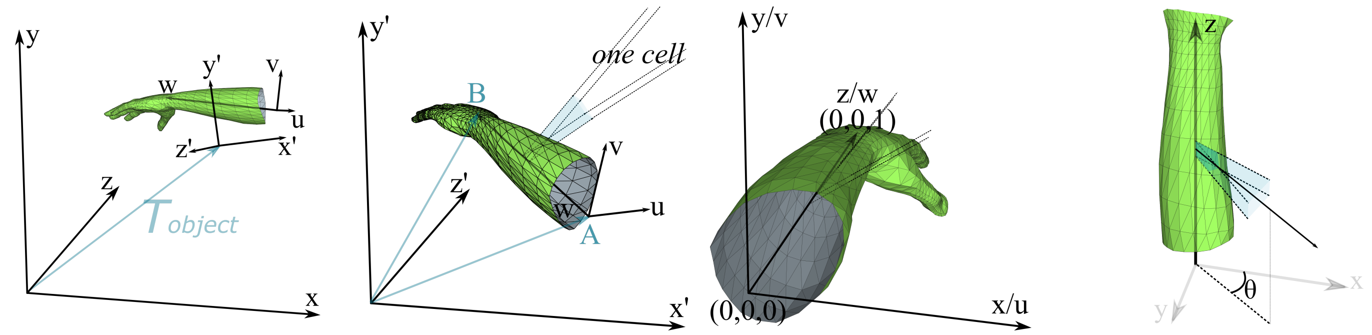
5.1 Rasterisation

Updating a field is a 2D rasteriastion problem, only with a non-traditional projection from 3D to 2D. The GPU's hardware rasteriser can be re-purposed for this. A dedicated shader updates the fields by rasterising the geometry into cells and computing a new distance value for each fragment. A set of indices are prepared with each one containing the vertex id and the field id to which it belongs. Vertices may appear in multiple fields, with a different relative position in each, through being referenced by multiple indices. We assign vertices to fields based on skinning weights.

For each field, Equation 1a is computed. This along with the indices and vertices are passed to the distance shader to begin updating the fields. The vertex shader transforms the vertices into field space (Section 4.2). A geometry shader transforms these into cylindrical coordinates (Section 4.2), performing wrapping and filtering as appropriate (Sections 5.1.1, 5.2.2), before passing them to the rasteriser, which renders the distances (Section 5.2).

This work is licensed under a Creative Commons Attribution 3.0 License. For more information, see http://creativecommons.org/licenses/by/3.0/.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2018.2859924, IEEE Transactions on Visualization and Computer Graphics



(a) World-space. T_{object} the transform of the object the field is defined relative to.

(b) Object-space. A & B are points defining the start and end of the field axis. u, v, w are basis vectors.

(c) Field-space. z has been normalised during change of basis.

(d) Cylindrical space

Fig. 1: Illustration of how the radial field exists and is parameterised within pertinent coordinate systems.

5.1.1 Wrapping

A complication of spherical coordinate systems is that unlike Cartesian systems, one or more axes will wrap around. While in most cases this does not present a problem, it may for the triangle filling algorithm. The rasteriser on a typical GPU is not designed to wrap around, and if a triangle crosses the 'seam' (an edge of the rasteriser's coordinate system) will rasterise in the wrong direction. To facilitate a wrap we use the technique described by Tarini [42], to whom we refer the reader for a more detailed explanation of this problem.

We say the device coordinates used by the rasteriser are (θ_1, z_1). These are θ and z, respectively as described in Section 4.2. We also add two additional parameters (θ_2, z_2), to store the coordinates on the cylinder. Typically, (θ_1, z_1) and (θ_2, z_2) will be identical. If a primitive crosses a seam however, we apply the following operations (Equation 4) to θ_1 and z_1 of all the primitive's coordinates.

$$\begin{aligned} \theta_1 &= frac(\theta_1 + 0.5) \\ \theta_2 &= frac(\theta_2 + 0.5) - 0.5 \end{aligned} \tag{4}$$

This has the effect of shifting all θ_1 by 180^o so that the hardware rasteriser will fill the triangles in the correct direction. θ_2 however has been transformed such that the coordinate system is now defined between -0.5 and 0.5, with the seam at 0. The hardware rasteriser correctly interpolates (θ_2, z_2) between negative and positive, so the final step is to check if $\theta_2 < 0$ in the fragment shader, and if so convert it back into $[0, 1]$ with the operation $\theta_2 = frac(\theta_2 + 1)$.

5.2 Distance Computation

The rasteriser will linearly interpolate the vertex parameters for each fragment. The distance cannot be interpolated this way however because in the embedded polar coordinate system it is equivalent to interpolating across the arc connecting the two sample points, not the straight line between them. Instead, for each fragment we compute the distance value using a ray-plane intersection. The test is performed in field space, with $o = (0, 0, z)$ and $dir = (x, y, 0)$. (x, y, z) being recovered as per Section 4.2. The origin and normal of the triangle plane are computed in the geometry shader and passed to each fragment.

5.2.1 Non-Convex Geometry

If multiple primitives overlap when viewed from the field axis, they will be projected into the same cell. We do not support this. To do so would make the method more complicated, while character meshes rarely have non star-convex geometry that must be represented in a single field. Instead we use depth-masking to store only the closest geometry. This is based on the assumption that all geometry represents the surface of the mesh, and so all but the nearest overlapping primitives must represent the surface of another element that will have its own field.

5.2.2 Quantisation Artefacts

One of the problems resulting from not interpolating distance values are artefacts resulting from quantisation in the ray-plane distance test. The quantisation in this case is not in the computation itself, but rather in the rasterisation. If a face is almost perpendicular to the axis normal, small changes as a result of quantisation to the resolution of the field can result in large distances. To avoid this we filter distances per-fragment based on the exact distances computed at the three face vertices.

If a triangle crosses the axis of the field, there is no way to rasterise it correctly and it is culled. This is done by the geometry shader by projecting the vertices into the XY plane in field space, then checking the face against the origin using edge functions. Since the origin is $(0, 0)$, we only have to compute the constant terms of the edge functions. Triangles that are back-facing or almost perpendicular are culled as well, based on a dot product with the axis.

5.3 Erasing the fields

The fields must be erased each frame. While they are overridden, there is no guarantee that a particular cell will be occupied from one frame to the next as the mesh deforms. If the field is not erased, old samples may be left behind.

6 Collision Detection and Response

The collision detection and response algorithm will depend on the physics simulation. We expect the main application to be cloth and particle simulations, so demonstrate

This work is licensed under a Creative Commons Attribution 3.0 License. For more information, see http://creativecommons.org/licenses/by/3.0/.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2018.2859924, IEEE Transactions on Visualization and Computer Graphics

particle collision detection and response. We designed our collision detection system to work with the cloth simulation described in Section 8. This is a position based dynamics system. That is, the collision detection system receives a set of current (q) and predicted (q_1) particle positions. The predicted positions being where the particles would be at the end of the time-step if there were no collisions. The collision detection system updates the predicted positions to account for any collisions, and returns them to the simulation. The response involves moving the positions in order to resolve any intersections. Other systems, such as force-based systems, may compute a force or impulse instead.

6.1 Broadphase

We implemented a broadphase stage where each radial field was approximated by a cylinder, with the radius set at design time. The broad-phase stage had no radial-field specific functionality and could be swapped out for any other implementation.

6.2 Pseudo-Continuous Collision Detection

Continuous Collision Detection (CCD) refers to techniques designed to detect collisions with sub-timestep accuracy in order to prevent artefacts such as tunnelling. CCD on distance fields is difficult because they contain no inherent information about the nature of the geometry. A naïve implementation would compute the motion of a particle relative to the field during a timestep, and check each intersecting cell. For non-trivial resolutions this is bandwidth intensive. Xu and Barbic [43] presented an algorithm accelerating CCD for distance fields using hierarchical data structures. The equivalent for our field would be a mip-map. These need to be computed ahead of time and so are not applicable here.

We perform Pseudo-Continuous Collision Detection (pCCD) by approximating the relative motion between a particle and a field. The vector (V_f) is defined in field space, from the current particle position (q) relative to the previous field transform, to the predicted particle position (q_1) relative to the current field transform (see Figure 2). This includes the motion of both the particle and the field. We walk the vector from start to end in n substeps, performing an intersection test at each point. The point of the first intersection is considered to be where the particle penetrates the field and is used for subsequent collision response computations. Another possibility would be to use a line-drawing algorithm in cylindrical space. The discretization artefacts that prevent our pCCD implementation from being true CCD come from the linearisation of rigid body rotations, and from marching the trajectory in discrete steps.

6.3 Collision Response

To detect collisions, we approximate the surface at a sample point in field space, and compute the signed point-plane distance. If the point is under the plane, it is intersecting. The normal of the surface approximation is used to compute the response.

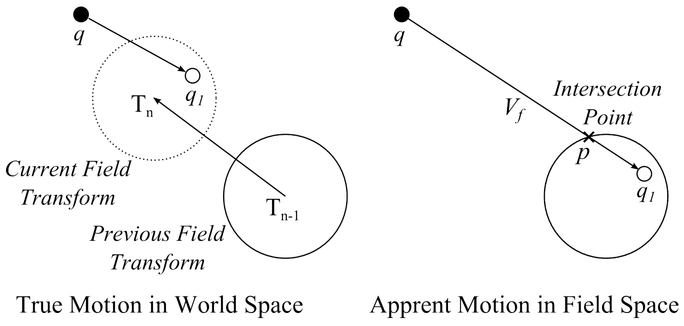


Fig. 2: pCCD is performed in field space. The relative particle positions include both rigid radial field transforms and particle motion.

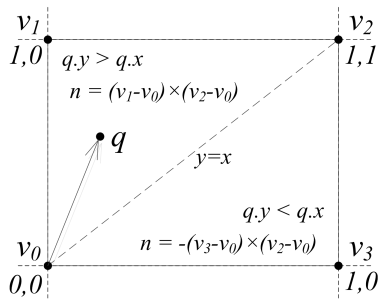


Fig. 3: Diagram of a radial field cell, showing how the normal is computed for each region, and how those regions are defined with respect to sample point q. v_0, v_1, v_2, v_4 are the positions of the sample points at the corners of the cell in whatever space the surface is being approximated. q is defined in cylindrical space relative to the cell origin.

6.3.1 *Surface Approximation*

Fuhrmann et al. [30] used a trilinear interpolation of 8 corner cells to approximate the surface at any location in their field. We approximate the surface by implicitly bisecting each cell into two triangles. The triangle containing the sample point is identified by comparing on which side of the line $x = y$ the point sits. The vertices for that triangle are retrieved and the plane computed from its edges, as shown in Figure 3. While character meshes are typically closed, on a per-field basis the surfaces may be open. To avoid introducing artefacts at the edges, if any of the vertices have a depth of zero the entire cell is considered empty.

The surface could be approximated in any space by transforming $v1, v2, v3, v4$ into that space (Section 4.2) before taking the cross-product. If the transformation is done in field space, the normal will be deformed when it is transformed back into world space however, due to the scale in the field space transform. We compute and pass a dedicated matrix to transform the normals to world space for the purposes of surface reconstruction: $T_{normalToWorldSpace}$ which is given by ${T_{toWorldSpace}}^{-1T}$. In theory, this is a requirement whenever normals are transformed in any system, but it is more likely to be of consequence in radial field implementations because they typically include non-uniform scales.

This work is licensed under a Creative Commons Attribution 3.0 License. For more information, see http://creativecommons.org/licenses/by/3.0/.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2018.2859924, IEEE Transactions on Visualization and Computer Graphics

6.3.2 Response

Our collision response is based on that of Fuhrmann et al. [30], modified slightly to support pCCD. The motion below surface (V_p) is determined by the pCCD stage (Equation 5, where p is the intersection point computed from the distance field). This is decomposed into components normal and tangential to the surface (Equations 5 & 7). The normal component V_n moves the particle to the surface, while the tangential component V_t emulates friction with coefficient μ.

V_t cannot be simply subtracted from the particle motion. This is because the penetration point p is approximate, with quantisation error introduced both in the pCCD stage and by the sampling of the field. V_t therefore may have a greater magnitude than V_q where $V_q = q_1 - q$, moving a particle farther than its predicted motion even when the collider is stationary. To avoid this, we calculate a new value p_{true} along V_q that is either (a) at the intersection of V_q with the surface, or (b) the original length of V_q if V_q does not intersect the surface (e.g. q starts below the surface) (Equation 9). With this new point we can calculate a V_t that is limited to the particle's true motion below the surface (Equation 10). The response is applied to q_1 to update the predicted position (Equation 11) and this is passed back to the cloth simulation.

$$V_p = q_1 - p \tag{5}$$

$$V_n = -n \left\langle V_p | n \right\rangle \tag{6}$$

$$V_t = \mu \cdot (V_p - V_n) \tag{7}$$

$$d_p = \frac{\langle n | o - q \rangle}{\left\langle n | \widehat{V_q} \right\rangle} \tag{8}$$

$$p_{true} = q + \widehat{V_q} \cdot max(0, min(d_p, \|V_q\|)) \tag{9}$$

$$V_t = u \cdot \widehat{V_t} \cdot \|(q_1 - p_{true}) - \langle q_1 - p_{true} | n \rangle \cdot n\| \tag{10}$$

$$q_1 = q_1 + V_n + V_t \tag{11}$$

As V_p includes the predicted particle motion, this approach both prevents penetrations and corrects existing ones. Where a particle crosses the surface during a timestep, the functionality is the same as the repulsion-based technique described by Bridson et al. [44] and Bender and Schmitt [45]. While we have adapted Fuhrmann et al.'s response for pCCD, there is nothing specific to radial fields and the response could be modified or replaced as necessary.

7 Implementation Details

We demonstrate the practicality of our technique by implementing it in Unity 5.4 and integrating it with a GPU accelerated cloth simulation. Our implementation uses compute shaders to bake skinning deformations into world space. The fields are cleared by compute shaders. We use the image-rasterisation pipeline, rather than our own algorithm in a compute shader, although that would also be feasible.

Our implementation targets SM 5.0. The shaders are written in HLSL within Unity's ShaderLab syntax. The field is stored in a 32-bit Compute Buffer and initialised to 2^{32} each frame. To perform the depth-masking described in Section 5.2.1, the buffer is written in the pixel shader using atomic InterlockedMin() operations. The buffer is bound as a UINT Unordered Access View. Since d is always positive, the floating point representation will interoperate with the atomic's integer comparators, so InterlockedMin() can operate on the field directly after calling asuint() on the depth values. The buffer is bound as a floating point resource to the Compute Shaders for collision detection and response.

8 Cloth Simulation

To demonstrate the utility of our technique, we integrated it into a cloth simulation system. Unity has cloth simulation abilities but they are limited and not accessible for modification. We therefore built a new cloth simulation system based on Position Based Dynamics (PBD) [46]. PBD systems operate on the positions of particles directly, rather than their forces or velocities. During a simulation step, particle positions are predicted independently based on external forces and inertia. The system then finds the nearest configuration that solves the constraints on the particles. In our system, collision constraints are then solved next. Finally new velocities are computed for each particle based on the change in position during the time-step. The approach is fast and unconditionally stable. Good surveys of PBD are available, for example Bender et al. [47], [48].

There are different approaches to find the configuration that best solves the constraints. We project each particle into a rest state for each constraint individually, based on constraint weights and strain [49]. We then perform "smart averaging" between them. This is based on the approach of Bouaziz et al. [50] and Weiler et al. [51]. Our cloth system operates entirely on the GPU using compute shaders.

Our cloth simulation is not as optimised as mature systems (e.g. NVidia's FleX). As can be seen in Section 6 however, the interface between the radial field system and the simulation is narrow and we would expect the technique to integrate easily with other particle-based simulations.

9 Performance Comparison

We compared our implementation with popular techniques for deformable object collision detection (Section 2.1). Three GPU-accelerated techniques were implemented based on the state-of-the-art and profiled against the radial fields implementation.

Real-time collision detection operates in two stages, a filtering stage in which potentially colliding primitives are enumerated (broadphase) and a second stage in which exact intersection tests are performed and the responses computed (narrowphase) [52]. The distribution of computational load between the them can vary dramatically depending on the data structure. For example, triangle based collision detection requires a point-triangle narrowphase test with a highly efficient broadphase, due to the typically high number of triangles. Distance fields support the narrowphase directly, and due to their low number can have very simple or no broadphase (Section 6.1).

This work is licensed under a Creative Commons Attribution 3.0 License. For more information, see http://creativecommons.org/licenses/by/3.0/.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2018.2859924, IEEE Transactions on Visualization and Computer Graphics

Since the stages cannot be decoupled when comparing heterogeneous narrowphases, we show the total collision detection and response times in Table 1. When considering point-triangle intersections, there are many trade-offs in robustness and performance. For the partitioning schemes (spatial hashing and BVH), we opted for a simple discrete point-triangle test, with a response similar to that described in 6.3.2. It is not robust enough to be used in practice, but it presents the best case scenario for these techniques in terms of performance.

9.1 Bounding Volume Hierarchies

Many works use GPUs to accelerate BVH construction [12], [14] and traversal [53]. We base our implementation on Lauterbach et al. [15] and Tang et al. [16], as these are concerned with deformable objects. We use agglomerative clustering [54], [55] to compute a binary-tree of AABBs. We then flatten this into an 8-ary tree [15] of uniform depth. On each frame, the AABBs are re-fitted. A single Compute Shader updates the whole tree bottom-up using for-loops, thread-masking and global memory synchronisation calls. Traversal is facilitated by work-queues based on Append/Consume buffers. For collision detection, *jobs* are created for every particle and the first node in the tree. A job consists of checking one particle against one volume. If penetrating, jobs are added for that particle and all child nodes, to be processed in a subsequent invocation. A single call processes all jobs for each level. The depth of the tree is known, so the CPU can make a fixed number of dispatch calls resulting in a queue that contains only leaf nodes. For these, a different shader performs vertex-triangle intersection tests and the collision response. Dispatch parameters are computed in Compute Shaders and used for indirect dispatch calls, allowing the implementation to run entirely on the GPU.

9.2 Spatial Hashing

Our spatial hashing implementation is based on that of Rumman et al. [19]. We use a triangle-parallel 26-separating computational voxelisation algorithm [56], [57] to write the spatial hash for each primitive on each frame. Rumman et al. perform only self-intersection and so can rely on temporal masking. We cannot make the same assumptions and so instead clear the cell-counts on each frame.

9.3 Distance Fields

Our Distance Field implementation is based on that of Fisher & Lin [4], updating a narrow band around an objects surface. We do this by computing an AABB around each primitive and performing a brute-force closest-point computation, such as for Yin et al's. [58] type-1 points. A depth-mask is used to select the closest distances for each cell, and it is cleared each frame. An ostensible optimisation would be to use voronoi regions such as prisms to update only changed cells. We attempted this but found that the volumes leaked, and are unaware of any works that have successfully taken this approach with deformable triangular meshes.

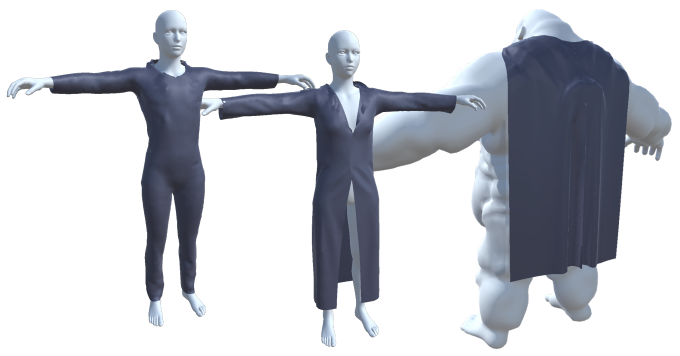


Fig. 4: Avatar and Cloth Models (not to scale) used during profiling. For each avatar Low, Medium and High detail versions were used.

9.4 Models and Configuration

The test models are shown in Figure 4. The models were optimised and subdivided to create configurations of varying complexity. Radial Fields, Distance Fields and Spatial Hashing had a spatial resolution of $1\,\mathrm{cm}$, with the exception of Distance Fields in conditions *9* & *10*, which had a resolution of $3.5\,\mathrm{cm}$. The Male and Female Human avatars were derived from the same base model and so have identical topologies, differing only in shape. Radial Fields and Distance Fields were applied per-bone, while Spatial Hashing and BVHs operate directly in avatar object or world space. Avatars were equipped with typical game bone rigs consisting of 50-60 bones. The Human models were $\sim 1.6\,\mathrm{m}$ tall, while the Ogre was $\sim 4\,\mathrm{m}$ tall.

Radial Fields and Distance fields were fitted to the bones automatically, such that they encompassed all vertices skinned to their bone. The spatial hash had a table size of 2564327 and a cell size of 20 based on manual tuning to avoid overflowing cells.

9.5 Profiling

Measurements were taken with the Unity Editor profiler and include only the time spent in functionality exclusive to the techniques. For example, skinning time was not included. Profiling was performed on a Windows 7 PC with a 3.4 GHz i7 and an NVidia GTX1080.

All tests used the same animation sequence, lasting ~10 seconds and consisting of some extreme locomotion and gymnastic motions that may be encountered in a typical 3D video game. For consistency and stability, extensive use of attachment points were made. These were implemented as soft constraints however, so all such particles still participated fully in collision detection and response. Our results are shown in Table 1.

10 Results

10.1 Computation Time

We consider total computation time to be the sum of the structure update and collision detection & response times. By this metric, we can see from Table 1 that radial fields match or exceed all other techniques. Radial fields offer an average speed-up of 27x compared to traditional distance

This work is licensed under a Creative Commons Attribution 3.0 License. For more information, see http://creativecommons.org/licenses/by/3.0/.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2018.2859924, IEEE Transactions on Visualization and Computer Graphics

fields with narrow-band updates, and 2.9x & 4.5x compared to per-primitive tests accelerated with BVHs and spatial hashing, respectively.

Each technique however has different dependencies, and therefore will scale differently. For example, Distance Fields and Radial Fields are highly dependent on the volume of the avatar and spatial resolution. This is because they are bound by the time to 'render' the cells, and the cell count is dependent on these parameters. This can be seen most clearly in Conditions 1 & 2 for Distance Fields, where a reduced number of triangles results in a longer computation time, as their larger size means a less efficient approximation of the narrow band. Distance Field's memory and computation times increase severely with resolution or volume. In Conditions 9 & 10 we had to reduce the spatial resolution of the distance field to $3.5\,\text{cm}$ in order to keep the simulation stable. As can be seen, radial fields are also sensitive to these parameters, but their demands increase at a lower rate. BVHs on the other hand depend only on the triangle count. This can be seen in Conditions 9 & 10 vs. 7 & 8, where the metric trajectories with respect to triangle count are inverted. We expect that BVHs would quickly begin to outperform the other techniques as spatial resolution increased. The proportion of the time spent clearing the Distance Field mask was $< 0.5\%$ of the total computation time in every condition.

Spatial Hashing should in theory have similar performance to Radial Fields with a constant offset for the point-triangle tests, since it voxelises only the surface with a subdivision resulting in a similar number of cells. In practice the performance is poorer. This is likely due to the 3D rasterisation being less efficient than the 2D rasterisation. Radial Fields take advantage of the GPU's hardware rasteriser which will have many optimisations, such as hierarchical region testing, that improve performance [59]. For interests sake, it took 4-5x longer to rasterise the faces of the model to the radial fields, than it took to render them to the screen.

All techniques run entirely on the GPU. Though with the exception of BVHs, all techniques require transforms to be extracted from the scene graph and sent to the GPU each frame. Radial Fields and Distance Fields, which are assigned per bone, are therefore more CPU intensive than Spatial Hashing (with only one pair of transforms) or BVHs (with none). We do not report these times because there is no technique specific functionality involved and the overhead is trivial.

10.2 Collision Detection

Our Distance Field implementation (Section 9.3) used a uniformly-sampled linear signed distance field, updated based on the AABBs of deforming primitives. That is, the entire grid was initialised even though only a narrow-band was updated each frame. This implementation, like Radial Fields, completely decouples the intersection tests from the underlying model - as can be seen from the collision detection times which are directly proportional to the particle counts. Spatial Hashing Collision Detection (CD) times depend on the probability of a hash collision, and therefore on model complexity, spatial resolution and table properties. BVHs CD depends on the tree itself and the number of particles, and so is loosely coupled to model complexity.

Distance Field intersection tests are the least computationally intensive and this is reflected in the CD times. They are the most bandwidth intensive however needing 8 samples per tri-linear interpolation for surface reconstruction. Radial Fields require fewer - 3 - memory reads per reconstruction, but converting the samples into Cartesian space is more expensive than Distance Field's direct interpolation. Both techniques spend less time in the CD stage than Spatial Hashing or BVHs. Radial Fields and Distance Fields, due to their regular access patterns, scale more predictably with particle count than do Spatial Hashing or BVHs. However, BVHs can cull large numbers of potential tests early on, resulting in improved performance in larger simulations. Compare BVH Condition 10 to 9, which is only fractionally longer despite processing 3x the number of particles.

10.3 Memory

Spatial Hashing is the most memory intensive technique. The table size and cell size must be sufficiently large to avoid cell overflows, or collisions may be missed. Expanding cells on demand is straightforward on the CPU, but not the GPU. The other techniques have comparable requirements. BVHs have an advantage with larger, but lower resolution models. For smaller models however the distance field based approaches outperform BVHs. The fields have many more elements than BVHs do nodes, but the nodes are larger. Distance fields scale poorly with volume. Readers should recall that we had to decrease the resolution for Distance Fields in Conditions 9 & 10. If they remained at $1\,\text{cm}$ like the other techniques, the fields would be over 124 Mb, compared to Radial Field's 3.16 Mb. While there are more efficient adaptive representations, these would prevent dynamic updates.

11 Discussion

11.1 Traditional Distance Fields

We expected distance fields to be outperformed, but were surprised by the extent. Since Fisher & Lin [4], authors have worked to improve initialisation times and memory requirements [2], [60], but with the exception of Gascon et al.'s [32] technique for tetrahedral meshes, there are no fundamentally new, real-time techniques for deforming them. We suggest this is due to the lack of a reliable voronoi technique for deforming triangular mesh models. While there are techniques for conservative-voxelisation [61], these typically involve over-scanning in some sense, meaning implementations must fall back on a depth mask. Techniques such as fast-marching [62] improve performance of the closest-point computation itself, but at a cost of accuracy.

11.2 Continuous Collision Detection

Collision detection techniques vary not only in performance, but also in feature-set. Spatial Hashing is one of the fastest techniques, but is one of the most difficult with which to support CCD. The band of voxels that are occupied is narrow compared to distance fields. A reliable implementation would need to rasterise the prism of a deforming triangle,

This work is licensed under a Creative Commons Attribution 3.0 License. For more information, see http://creativecommons.org/licenses/by/3.0/.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2018.2859924, IEEE Transactions on Visualization and Computer Graphics

					GPU Computation Times (ms) and **Memory (Mb)**											
		Models			Radial Fields			Distance Fields			Spatial Hashing			BVHs		
C.	Avatar	Cloth	Triangles	Particles	Structure		CD	Structure		CD	Structure		CD	Structure		CD
					Mem.	Upd.		Mem.	Upd.		Mem.	Upd.		Mem.	Upd.	
1	Male Low	Trouser & Top Low	10827	1297	**0.21**	0.06	0.04	**2.37**	5.18	0.03	**205.42**	0.93	0.08	**0.95**	0.03	0.19
2	Male Low	Trouser & Top Med	10827	4825	**0.21**	0.07	0.09	**2.37**	5.18	0.05	**205.42**	0.86	0.12	**0.95**	0.03	0.34
3	Male Med	Trouser & Top Low	37744	1297	**0.21**	0.06	0.04	**1.31**	2.60	0.02	**205.42**	0.26	0.11	**3.96**	0.11	0.24
4	Male Med	Trouser & Top Med	37744	4825	**0.21**	0.06	0.08	**1.31**	2.54	0.04	**205.42**	0.26	0.16	**3.96**	0.10	0.42
5	Female Med	Overcoat Med	37744	5118	**0.21**	0.05	0.07	**1.01**	2.95	0.06	**205.42**	0.23	0.11	**3.96**	0.10	0.32
6	Female Med	Overcoat High	37744	13090	**0.21**	0.05	0.16	**1.01**	2.94	0.11	**205.42**	0.23	0.14	**3.96**	0.09	0.61
7	Female High	Overcoat Med	64176	5118	**0.20**	0.08	0.07	**1.31**	3.83	0.05	**205.42**	0.23	0.14	**6.55**	0.15	0.36
8	Female High	Overcoat High	64176	13090	**0.20**	0.08	0.17	**1.31**	3.84	0.12	**205.42**	0.23	0.14	**6.55**	0.15	0.75
9	Ogre	Cape Med	17211	8562	**3.16**	0.10	0.11	***3.01***	*9.35*	*0.07*	**205.42**	1.74	0.10	**1.38**	0.04	0.27
10	Ogre	Cape High	17211	29737	**3.16**	0.11	0.31	***3.01***	*9.20*	*0.21*	**205.42**	1.77	0.22	**1.38**	0.03	0.49

TABLE 1: Memory requirements (Mem.) and execution times of the structure update (Upd.) and collision detection & response (CD), for the 10 test conditions (*C.*).

and then march along the particle motion through the grid. BVHs can perform continuous intersection tests against their volumes, and the underlying primitives in one step. Of the techniques that require marching, radial fields have the best approximation because they implicitly delineate the object interior, reducing the likelihood of tunneling.

11.3 Ease of Implementation

Ease of implementation is subjective, but broadly, the number of heterogeneous functionalities required to implement radial fields is lower than BVHs but higher than distance fields and equivalent to spatial hashing. The unambiguous delineation of the mesh interior is a significant benefit when writing collision detection algorithms. Narrow-band distance fields are the only other technique to offer this, and they do so with caveats - for example, samples may become outdated. With regards to tunable parameters, Radial Fields and Distance Fields have only resolution to select, which is physically based. BVHs have no tunable parameters. Spatial Hashing does not need to be configured per-bone like Radial Fields and Distance Fields, but requires deeper knowledge in order to choose optimal table parameters.

11.4 Limitations

Radial fields trade off generality of techniques such as hp-adaptive distance fields [35], in favour of GPU acceleration, but this imposes limitations. Radial fields can only support star-convex geometry. For some elements, such as non-articulated hands, or a character's ears, a single field could approximate a convex hull. In most cases though overlapping geometry must be covered with multiple fields. Radial fields introduce quantisation noise. This is dependent on the resolution of the field. Previous analyses of distance field fidelity are directly applicable here. Radial fields, like other grid based structures, have difficulty with continuous collision detection. We present pCCD, which does not guarantee robustness. A robust CCD implementation is possible with a marching algorithm that visits every cell, but at a cost of performance.

12 Future Works

Radial field implementations are complicated by workarounds for the vulnerability in the ray-plane distance test to rasterisation quantisation error. An alternative to the ray-plane distance test would be to compute the distance using the polar straight line equation, which can be derived from the Cartesian line equation ($y = mx + c$) and Equation 3. The coefficients for this could be computed in the geometry shader and passed to the fragment shader. However, because the triangles are defined by three edges, the system would need to interpolate between three sets of coefficients, and how this would be done is not clear.

13 Conclusion

We have presented Radial Fields, distance fields parameterised in 2D cylindrical space rather than 3D Cartesian space. Radial fields are more application-specific than traditional distance fields, but for these applications they offer substantial benefits. We profiled our radial field implementation across a number of models, demonstrating sub-millisecond computation, collision detection and response times for simulations with a range of complexities.

Radial fields outperform traditional distance fields by over an order of magnitude. They also outperform triangle-based tests with broadphase stages based on the state-of-the-art in GPU-based spatial hashing and bounding volume hierarchies. Radial fields scale differently to these techniques however, and which is most suitable will depend much on the application. Radial fields use the hardware rasteriser of the GPU, making them easy to implement by taking advantage of the existing, highly optimised, rasterisation pipeline. Radial fields implicitly define the mesh interior, making them a cost-effective way of describing volumes. This simplifies collision detection and response algorithms, and makes them more robust to large time-steps. While the implicit volumetric representation reduces the likelihood of tunneling compared to discrete triangle-based tests, achieving robustness comparable to continuous triangle-based tests could become very bandwidth intensive depending on field resolution.

To demonstrate the practicality of radial fields in real applications, we created our test implementation in Unity 5.4 and integrated it with a position-based dynamics cloth simulation. In the future, radial fields could be improved by replacing the ray-plane distance test in the rasterisation stage with the polar straight line equation, making implementations more robust to rasterisation quantisation and even simpler to implement.

This work is licensed under a Creative Commons Attribution 3.0 License. For more information, see http://creativecommons.org/licenses/by/3.0/.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2018.2859924, IEEE Transactions on Visualization and Computer Graphics

REFERENCES

[1] Ericson, *Real-Time Collision Detection*. CRC Press, 2004.

[2] Liu and Kim, "Exact and Adaptive Signed Distance Fields Computation for Rigid and Deformable Models on GPUs," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, no. 5, pp. 714–725, 5 2014.

[3] Sud, Govindaraju, Gayle, and Manocha, "Interactive 3D distance field computation using linear factorization," in *Proceedings of the 2006 symposium on Interactive 3D graphics and games - SI3D '06*. New York, USA: ACM Press, 2006, p. 117.

[4] Fisher and Lin, "Fast penetration depth estimation for elastic bodies using deformed distance fields," in *Proceedings of the Eurographic Workshop on Computer Animation and Simulation*. Springer-Verlag New York, Inc., 2001, pp. 99–111.

[5] Teschner, Kimmerle, Heidelberger, Zachmann, Raghupathi, Fuhrmann, Cani, Faure, Magnenat-Thalmann, Strasser, and Volino, "Collision Detection for Deformable Objects," *Computer Graphics Forum*, vol. 24, no. 1, pp. 61–81, 3 2005.

[6] Zachmann and Langetepe, "Geometric Data Structures for Computer Graphics," in *Siggraph 2003 Course Notes, Tutorial 16*, 2003.

[7] Larsson and Akenine-Möller, "A dynamic bounding volume hierarchy for generalized collision detection," *Computers and Graphics (Pergamon)*, vol. 30, no. 3, pp. 451–460, 2006.

[8] Wald, "On fast Construction of SAH-based Bounding Volume Hierarchies," in *2007 IEEE Symposium on Interactive Ray Tracing*, vol. 1. IEEE, 9 2007, pp. 33–40.

[9] Mujika, Oyarzun, and Arrieta, "Real time accurate collision detection for virtual characters," in *WSCG 2010: Communication Papers Proceedings: 18th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision in co-operation with EUROGRAPHICS*, 2010, pp. 123–130.

[10] Redon, Lin, Manocha, and Kim, "Fast Continuous Collision Detection for Articulated Models," *Journal of Computing and Information Science in Engineering*, vol. 5, no. 2, p. 126, 2005.

[11] Karras and Aila, "Fast parallel construction of high-quality bounding volume hierarchies," in *Proceedings of the 5th High-Performance Graphics Conference - HPG '13*. New York, USA: ACM Press, 2013.

[12] Lauterbach, Garland, Sengupta, Luebke, and Manocha, "Fast BVH Construction on GPUs," *Computer Graphics Forum*, vol. 28, no. 2, pp. 375–384, 4 2009.

[13] He, Ortiz, Enquobahrie, and Manocha, "Interactive Continuous Collision Detection for Topology Changing Models Using Dynamic Clustering," in *Proceedings of the 19th Symposium on Interactive 3D Graphics and Games - i3D '15*. New York, USA: ACM Press, 2015, pp. 47–54.

[14] Meister and Bittner, "Parallel BVH construction using k-means clustering," *The Visual Computer*, vol. 32, no. 6-8, pp. 977–987, 2016.

[15] Lauterbach, Mo, and Manocha, "GProximity: Hierarchical GPU-based operations for collision and distance queries," *Computer Graphics Forum*, vol. 29, no. 2, pp. 419–428, 2010.

[16] Tang, Manocha, Lin, and Tong, "Collision-Streams: Fast GPU-based Collision Detection for Deformable Models," in *Symposium on Interactive 3D Graphics and Games - I3D '11*, vol. 1, no. 212. New York, USA: ACM Press, 2011, p. 63.

[17] Teschner, Heidelberger, Müller, Pomeranets, and Gross, "Optimized Spatial Hashing for Collision Detection of Deformable Objects," in *Proceedings of Vision, Modeling, Visualization VMV03*, 2003, pp. 47–54.

[18] Wong, Leach, and Zambetta, "An adaptive octree grid for GPU-based collision detection of deformable objects," *The Visual Computer*, vol. 30, no. 6-8, pp. 729–738, 6 2014.

[19] Rumman, Schaerf, and Bechmann, "Collision detection for articulated deformable characters," *Proceedings of the 8th ACM SIGGRAPH Conference on Motion in Games - SA '15*, pp. 215–220, 2015.

[20] Faure, Barbier, Allard, and Falipou, "Image-based Collision Detection and Response between Arbitrary Volumetric Objects," in *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, Dublin, Ireland, 2008, pp. 1–8.

[21] Jang and Han, "Fast collision detection using the A-buffer," *The Visual Computer*, vol. 24, no. 7-9, pp. 659–667, 7 2008.

[22] Rodriguez-Navarro, Sainz, and Susin, "GPU Based cloth simulation with Moving Humanoids," in *Proc. XV Congreso Español de Informática Gráfica (CEIG2005)*, 2005, pp. 147–155.

[23] Vassilev, Spanlang, and Chrysanthou, "Fast Cloth Animation on Walking Avatars," *Computer Graphics Forum*, vol. 20, no. 3, pp. 260–267, 9 2001.

[24] Gibson, "Using distance maps for accurate surface representation in sampled volumes," in *Proceedings of the 1998 IEEE symposium on Volume visualization - VVS '98*, vol. 6. New York, USA: ACM Press, 1998, pp. 23–30.

[25] Jones and Satherley, "Using distance fields for object representation and rendering," in *Proc. 19th Ann. Conf. of Eurographics (UK Chapter)*, 2001, pp. 37–44.

[26] Jones, Bærentzen, and Sramek, "3D distance fields: A survey of techniques and applications," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 4, pp. 581–599, 2006.

[27] Frisken and Perry, "Designing with distance fields," in *International Conference on Shape Modeling and Applications 2005 (SMI' 05)*, 2005, pp. 58–59.

[28] Weller and Zachmann, "Inner Sphere Trees for Proximity and Penetration Queries," in *2009 Robotics: Science and Systems Conference (RSS)*. Seattle, WA, USA: Springer Vienna, 2011.

[29] Weller, Frese, and Zachmann, "Parallel Collision Detection in Constant Time," in *Workshop on Virtual Reality Interaction and Physical Simulation VRIPHYS (2013)*, 2013.

[30] Fuhrmann, Sobottka, and Groß, "Distance Fields for Rapid Collision Detection in Physically Based Modeling," in *Proceedings of GraphiCon 2003*, 2003, p. 5865.

[31] Pantaleoni, "VoxelPipe: A Programmable Pipeline for 3D Voxelization," in *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics - HPG '11*, vol. 1. New York, USA: ACM Press, 2011, pp. 99–106.

[32] Gascon, Espadero, Perez, Torres, and Otaduy, "Fast deformation of volume data using tetrahedral mesh rasterization," in *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation - SCA '13*. New York, USA: ACM Press, 2013, p. 181.

[33] ElBadrawy, Hemayed, and Fayek, "Rapid collision detection for deformable objects using inclusion-fields applied to cloth simulation," *Journal of Advanced Research*, vol. 3, no. 3, pp. 245–252, 7 2012.

[34] McAdams, Zhu, Selle, Empey, Tamstorf, Teran, and Sifakis, "Efficient elasticity for character skinning with contact and collisions," *ACM Transactions on Graphics*, vol. 30, no. 4, p. 1, 7 2011.

[35] Koschier, Deul, Brand, and Bender, "An hp-Adaptive Discretization Algorithm for Signed Distance Field Generation," *IEEE Transactions on Visualization and Computer Graphics*, vol. 23, no. 10, pp. 2208–2221, 2017.

[36] Funfzig, Ullrich, and Fellner, "Hierarchical spherical distance fields for collision detection," *IEEE Computer Graphics and Applications*, vol. 26, no. 1, pp. 64–74, 1 2006.

[37] Wong, Lin, Hung, Huang, and Lii, "Radial view based culling for continuous self-collision detection of skeletal models," *ACM Transactions on Graphics*, vol. 32, no. 4, p. 1, 7 2013.

[38] Carr, Beatson, Cherrie, Mitchell, Fright, McCallum, and Evans, "Reconstruction and representation of 3D objects with radial basis functions," in *Proceedings of the 28th annual conference on Computer graphics and interactive techniques - SIGGRAPH '01*. New York, USA: ACM Press, 2001, pp. 67–76.

[39] Moustakas, Tzovaras, and Strintzis, "SQ-Map: Efficient Layered Collision Detection and Haptic Rendering," *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, no. 1, pp. 80–93, 1 2007.

[40] Vogiannou, Strintzis, Moustakas, and Tzovaras, "Distance Maps for Collision Detection of Deformable Models," in *MCCSIS'08 - IADIS Multi Conference on Computer Science and Information Systems*, 2008, pp. 216–220.

[41] Kurihara, Anjyo, and Thalmann, "Hair Animation with Collision Detection," in *Models and Techniques in Computer Animation*. Tokyo: Springer Japan, 1993, pp. 128–138.

[42] Tarini, "Cylindrical and Toroidal Parameterizations Without Vertex Seams," *Journal of Graphics Tools*, vol. 16, no. 3, pp. 144–150, 2012.

[43] Xu and Barbič, "Continuous Collision Detection Between Points and Signed Distance Fields," in *Workshop on Virtual Reality Interaction and Physical Simulation VRIPHYS 2014*, 2014.

[44] Bridson, Fedkiw, and Anderson, "Robust treatment of collisions, contact and friction for cloth animation," *ACM Transactions on Graphics*, vol. 21, no. 3, pp. 594–603, 7 2002.

[45] Bender and Schmitt, "Constraint-based collision and contact handling using impulses," in *Proceedings of the 19th International Conference on Computer Animation and Social Agents*, 2006, pp. 3–11.

[46] Müller, Heidelberger, Hennix, and Ratcliff, "Position based dynamics," *Journal of Visual Communication and Image Representation*, vol. 18, no. 2, pp. 109–118, 4 2007.

This work is licensed under a Creative Commons Attribution 3.0 License. For more information, see http://creativecommons.org/licenses/by/3.0/.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2018.2859924, IEEE Transactions on Visualization and Computer Graphics

[47] Bender, Müller, Otaduy, and Teschner, "Position-based Methods for the Simulation of Solid Objects in Computer Graphics," in *STAR Proceedings of Eurographics 2013*, 2013.

[48] Bender, Müller, Otaduy, Teschner, and Macklin, "A Survey on Position-Based Simulation Methods in Computer Graphics," *Computer Graphics Forum*, vol. 33, no. 6, pp. 228–251, 9 2014.

[49] Macklin, Müller, and Chentanez, "XPBD," in *Proceedings of the 9th International Conference on Motion in Games - MIG '16*. New York, USA: ACM Press, 2016, pp. 49–54.

[50] Bouaziz, Martin, Liu, Kavan, and Pauly, "Projective dynamics," *ACM Transactions on Graphics*, vol. 33, no. 4, pp. 1–11, 7 2014.

[51] Weiler, Koschier, and Bender, "Projective fluids," in *Proceedings of the 9th International Conference on Motion in Games - MIG '16*. New York, USA: ACM Press, 2016, pp. 79–84.

[52] Weller, "A Brief Overview of Collision Detection," in *New Geometric Data Structures for Collision Detection and Haptics*, ser. Springer Series on Touch and Haptic Systems. Heidelberg: Springer International Publishing, 2013, pp. 9–47.

[53] Vinkler, Havran, and Bittner, "Performance Comparison of Bounding Volume Hierarchies and Kd-Trees for GPU Ray Tracing," *Computer Graphics Forum*, vol. 35, no. 8, pp. 68–79, 12 2016.

[54] Walter, Bala, Kulkarni, and Pingali, "Fast agglomerative clustering for rendering," *RT'08 - IEEE/EG Symposium on Interactive Ray Tracing 2008, Proceedings*, pp. 81–86, 2008.

[55] Gu, He, Fatahalian, and Blelloch, "Efficient BVH construction via approximate agglomerative clustering," in *Proceedings of the 5th High-Performance Graphics Conference - HPG '13*. New York, New York, USA: ACM Press, 2013, p. 81.

[56] Rauwendaal and Bailey, "Hybrid computational voxelization using the graphics pipeline," *Journal of Computer Graphics Techniques*, vol. 2, no. 1, pp. 15–37, 2012.

[57] Schwarz and Seidel, "Fast parallel surface and solid voxelization on GPUs," *ACM Transactions on Graphics*, vol. 29, no. 6, p. 1, 2010.

[58] Yin, Liu, and Wu, "Fast Computing Adaptively Sampled Distance Field on GPU," *Pacific Graphics 2011*, pp. 25–30, 2011.

[59] Davidovič, Engelhardt, Georgiev, Slusallek, and Dachsbacher, "3D Rasterization: A Bridge Between Rasterization and Ray Casting," in *Proceedings of Graphics Interface 2012*, 2012, pp. 201–208.

[60] Park, Lee, Kim, and Kim, "CUDA-based Signed Distance Field Calculation for Adaptive Grids," in *2010 10th IEEE International Conference on Computer and Information Technology*, no. May. IEEE, 6 2010, pp. 1202–1206.

[61] Hasselgren, Akenine-Möller, and Ohlsson, "Conservative Rasterization," in *GPU Gems 2*. Reading, MA: Addison-Wesley, 2005, pp. 677–690.

[62] Marchal, Aubert, and Chaillou, "Collision between deformable objects using fast-marching on tetrahedral models," *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation - SCA '04*, vol. i, no. January, p. 121, 2004.

This work is licensed under a Creative Commons Attribution 3.0 License. For more information, see http://creativecommons.org/licenses/by/3.0/.