Nucleus: Towards a Unified Dynamics Solver for Computer Graphics

Jos Stam
Autodesk, Inc.
Jos.Stam@autodesk.com

Abstract

This paper presents a unified dynamics solver developed by the author which was first released in Autodesk™ MAYA 8.5. The solver however is a standalone library which could potentially be used in other applications. Current dynamics solvers are usually fine tuned for specific effects such as rigid bodies or cloth. Handling the interaction between these solvers is often problematic as one of them takes precedence over the others. In our Nucleus solver we model all matter as a simplicial complex: a generalization of a triangle mesh that also includes points, curves and solids. This allows interactions such as collisions between various elements of different dimensionality. The internal deformations such as stretch and bend are handled through constraints instead of springs. This makes the simulation more stable for stiff materials such as cloth. Through mutual interactions and constraints many interesting phenomena emerge automatically. The basic philosophy behind Nucleus is that complexity arises by combining simple constraints.

1. Introduction

The convincing simulation of interacting deformable objects is hard to achieve using traditional animation techniques such as key-framing alone. Therefore there is a need in computer graphics to rely on physics-based dynamics solvers. Instead of specifying exact poses through key frames an animator specifies material properties of the object's and external forces. Given this information the dynamics solver then ideally computes snapshots of the states of all the objects over fixed time-steps. Most current solvers are fine tuned for a specific effect such as rigid bodies and cloth. Resolving interactions between such solvers can become problematic. For example, imagine a rigid body like a soccer ball being kicked in a goal. There will be a two way interaction between the ball and the net. Achieving this effect by connecting a rigid body solver to a curve-based solver for the goal net can be problematic. In this paper we present a solver that tries to resolve these interactions simultaneously.

We describe both how we model different shapes of matter and how we simulate them. We decided to use a simplicial complex as our shape model as it includes points, curves, surfaces and solids in a unified framework. For the simulation part we use a space-time based approach for the collisions and a constraint based approach to account for deformations. This approach results in simulations that are relatively stable for stiff materials such as cloth.

By allowing various elements of matter to interact in this manner we get interesting emergent behaviors. Even though each interaction is simple more complex behaviors emerge. For example a flapping flag can be simulated using a simple directional wind field and an inextensible piece of cloth. The flapping behavior emerges from the drag and lift constraints battling the stretch constraint. The behavior emerges without the need for a complicated air flow model. Throughout our research we emphasize simplicity as it vastly reduces the amount of code and consequently the amount of potential bugs. This is not just an aesthetic bias on our part rooted in a desire to achieve mathematical elegance. In practice adhering to this principle results in more robust and stable commercial products.

2. Previous and Related Work and the History of Nucleus

Simulations based on physics are evidently not novel in computer graphics. This approach was pioneered in the late eighties by many researchers [18]. Early work focused mainly on spring-based models for deformable matter and used either explicit or implicit methods. In addition many works have focused on simulating rigid bodies [5,13,17]. As far as we know the first paper to handle deformations as constraints was Provot's strain limiting procedure used for cloth [16], see also [9] and the pioneering work of Moreau [14]. Since then others have used this approach [12,15] to good use. Recently this approach seems to gain acceptance in academia as well [10].

The system closest in spirit to ours is the position based dynamics work of Müller et al. [15] developed independently from us and the collision work by Bridson et al. [6]. We do not cover here in detail the vast literature on this subject but only cite work that directly inspired us or is closely related. We refer the reader instead to one of the many surveys on this topic and the references in the papers cited. The goal of this paper is to give an overview of the ideas and techniques that were implemented in the Nucleus library.

Research on the Nucleus solver started as a small project by the author of this paper in the fall of 2000 in Seattle to create a very simple cloth solver for a cool live demo of smoke interacting with cloth. The idea was to replace springs with hard links. Since cloth is not stretchy it seemed like a bad idea to use very stiff springs to model it. So instead we started with hard links treated like constraints. Stiff springs have several problems. Explicit integrators require small time steps to achieve stability which result in long simulation times. On the other hand stable implicit time integration schemes damp out off spring motion which results in overly damped animations. We will make these points more concrete in a simple setting below.

The system we implemented initially was so simple that we wrote a version for the Palm just for fun back in 2001 for a very small 8x8 piece of cloth, followed by an implementation on the PocketPC with variable resolution. Both demos were "beamed" around at that time. We also showed a demo of cloth interacting with smoke in the "Visual Simulation of Smoke" paper presentation at SIGGRAPH in Los Angeles in 2001.

But it wasn't until the author of this paper showed the demos at the SIGGRAPH 2003 annual Alias|wavefront's user's group in San Diego that some buzz was generated amongst our user base. Subsequently we were asked by upper management at Alias|wavefront to replace the existing cloth solver in MAYA with our new one.

This task was quite a challenge since the existing cloth solver in MAYA was pretty sophisticated. Soon after we got seriously involved in this project we realized that our framework could accommodate other elements than cloth. That happened somewhere in the summer of 2004 and that was when the concept of a Nucleus solver really took off. After that we wrote many prototypes and the final version of the solver was written in the (hot) fall and (early) winter of 2005 in Toronto, Canada. It was then further refined and integrated into MAYA during 2006.

We made sure to build an API around our solver such that any changes at a low level would not affect function calls on the MAYA side. Our Nucleus solver is tiny compared to the MAYA source code: about a meager 100 files versus the 40,000 or so files populating the MAYA code base.

The capabilities of the new solver were first demonstrated at Autodesk's user's group at SIGGRAPH 2006 by Duncan Brinsmead in Boston in July and later in a key note talk by the author at EUROGRAPHICS 2006 in September in Vienna. The solver was first released in early 2007 in Autodesk™ MAYA 8.5 Unlimited with the release of nCloth. This release only exposed the cloth capabilities. In 2008 we released an nParticle feature allowing the interaction between particles and nCloth objects. We are hoping to add many more features in the future and enhance the Nucleus solver.

3. Shape Model: Simplicial Complexes

Since we are interested in modeling a whole range of shapes in a unified manner we decided to use the theory of simplicial complexes. It is a well known fact that any surface can be approximated by a triangular mesh to any arbitrary precision. A generalization of this result is a theorem first proved by Brouwer in 1910 which loosely states that "Every continuous mapping can be approximated by a piece-wise linear simplicial map." Or more precisely in math-speak:

Theorem:
Let K and L be complexes; let K be finite.
Given a continuous map $h : |K| \rightarrow |L|$
there is an N such that h has a simplicial approximation $f : \mathrm{sd}^N : \mathrm{K} \rightarrow \mathrm{L}$.

For our purposes it is sufficient to define a simplicial complex as an assemblage of simplices. A simplex is a generalization of a triangle to any dimension. Figure 1 shows four different k-simplices that model points, edges, triangles and tetrahedra. More complicated shapes are modeled by gluing these building blocks together. The blocks do not need to have the same dimension as shown in Figure 2. The definition of a simplicial complex is purely topological as it establishes a relationship between elements of an arbitrary set. The latter can be points in space, masses, colors, etc. We therefore neatly separate the topology from the geometry. In practice our implementation of the simplicial complex code contains only `int`s and no `float`s for example.

Figure 1: four k-simplices.

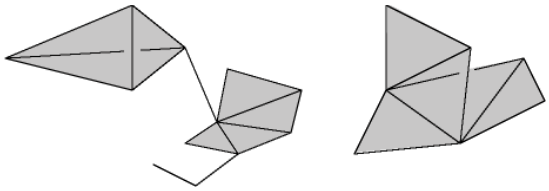

Figure 2: Two examples of a simplicial complex.

The neat aspect of using a unified model for the shapes is that it leads to a very elegant implementation using only a single data structure:

```
class simplex {
   int k;
   int sign;
   int vertex[k+1];
   int child[k+1];
   int n_parents;
   int parent[n_parents];
};
```

For each simplex we store its dimension k. We also store the k+1 indices of the elements and the k+1 (k-1)-simplices that it contains. For example a 2-simplex (triangle) contains 3 1-simplices (edges). A simplex can have an arbitrary number of parents. For example, a point (0-simplex) in a mesh can have an arbitrary number of incident edges (1-simplices). In this case the number of parents of the point is commonly called its valence. We also store a sign for each simplex for the following reason. For many operations it helps if the indices of the elements are stored in lexicographical order. However, this rearrangement of the indices can change the orientation of the simplex. When the number of transpositions is odd the sign is -1 otherwise it is 1. A zero sign value indicates that the element does not have an orientation. The sign also allows many algebraic operations to be simplified on simplices.

In this paper we will not get into all the details of implementing operations on simplical complexes. However, the above data structure allows us to implement many queries effortlessly that are needed to set up the constraints described below.

We encourage the reader who is interested in learning more about this topic to consult the excellent introduction to this topic by Alexandrov [1]. Also we recommend his more detailed monograph which has proven to be very helpful [2]. Both books are also very affordable because they have been cheaply reprinted by Dover publication.

4. Dynamics

4.1. Basic Equations

The dynamics of a simplicial complex is defined by the motion of its N vertices. We can compact this description in a $3N$ vector:

$$\mathbf{x}(t) = \big(x_1(t), \dots, x_N(t)\big).$$

The particles evolve due to external forces and internal deformations defined by the simplices and other factors as explained below. The laws that govern the motion of the particles are well known since Newton stated them in his famous *Principia* in 1687. In particular his second law states that (assuming unit masses):

$$\begin{aligned} \ddot{\mathbf{x}}(t) &= -\nabla f(\mathbf{x}) + \mathbf{f}_e \\ \mathbf{x}(0) &= \mathbf{x}_0 \\ \dot{\mathbf{x}}(0) &= \mathbf{v}_0 \end{aligned}$$

where $f(x)$ is the internal energy due to deformations and $\mathbf{f}_e$ model external forces like gravity. The initial state is defined by the initial positions and velocities of the particles. An alternative way to specify the dynamics is to require that the particles minimize the total energy at each instant of time:

$$E = \frac{1}{2}\,|\dot{\mathbf{x}}|^2 + f(\mathbf{x}) + \mathbf{f}_e \cdot \mathbf{x}$$

The first term is the total kinetic energy, the second term is the potential energy and the last term is the work done by the external forces. A very good introduction from a mathematical point of view is the monograph by Arnold [3]. These equations have been around for over 300 years and one would expect that there is a standard numerical procedure to solve them. However, this is not the case and to understand the difficulties we turn to a simple problem in more detail.

4.2. On the Motion of a Simple Spring

It is interesting that even the problem of solving the dynamics of a simple linear spring exhibits the behavior and difficulties common to more sophisticated solvers. This section is not a thorough overview of numerical integrators. For a good review see the excellent book by the group from my alma mater at l'Université de Genève [11]. The intent here is to focus on one of the simplest problems and understand the basic numerical problems one can encounter.

The equations for a linear spring are

$$\ddot{x}(t) = -x(t)$$
$$x(0) = x_0$$
$$v(0) = \dot{x}(0) = v_0.$$

To visualize the motion of the spring we can draw its trajectory in the phase space (x, v), a plane in this case. From the conservation of energy:

$$E = \frac{1}{2} v^2 + \frac{1}{2} x^2 = \frac{1}{2} v_0^2 + \frac{1}{2} x_0^2$$

we know that the trajectories are circles in the phase plane whose radius is a function of the initial state as shown in Figure 3. The equation can also be computed analytically in this case. There is an elegant way to obtain this result by introducing the complex number $z = x + iv$. In this manner the equation for the motion of the spring reduces to an ordinary differential equation:

$$\dot{z}(t) = -iz(t)$$
$$z(0) = z_0$$

Whose solution is $z(t) = z_0 e^{-it}$. This proves that the motion proceeds clock-wise along the trajectories. The equation in the complex domain also shows that the trajectories are tangent to the vector field as shown in Figure 4.

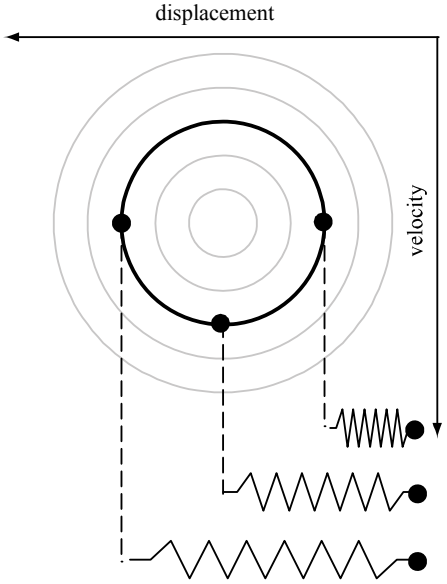

Figure 3: Trajectories of the spring in phase space.

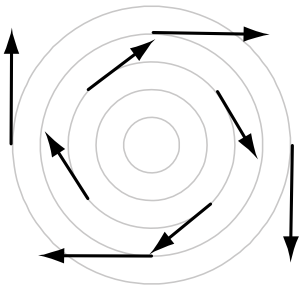

Figure 4: Trajectory of the spring is tangent to the vector field.

We now analyze three methods to solve this equation numerically with a fixed time step h. The time derivative between two consecutive states is approximated by:

$$\dot{z}(t) \approx \frac{z^1 - z^0}{h}.$$

In an explicit scheme the right hand side of the equation is evaluated at the current state, so that

$$z^1 = (1 + h)^2 e^{-ih} z^0.$$

In an implicit scheme on the other hand the right hand side of the equation is evaluated at the next state which results in the following update

$$z^1 = (1 + h)^{-2} e^{-ih} z^0.$$

We see that in an explicit scheme the motion of the spring is an outward spiral. This means that it gains energy over time and is thus inherently unstable. This is undesirable in general. The implicit scheme on the other hand is unconditionally stable by dissipating energy and the motion is that of an inward spiral. The problem with implicit methods is that there is no direct control over the amount of dissipation which depends on the time step. Figure 5 depicts this situation.

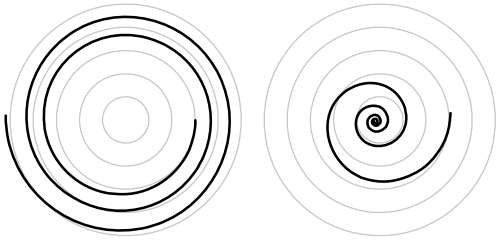

Figure 5: Trajectory of the spring in phase space using explicit integration (left) and implicit integration (right).

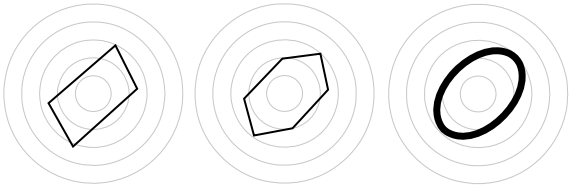

Figure 6: Symplectic trajectories for h=sqrt(2), 1 and 0.5.

A natural alternative is to combine the two schemes hoping that the dissipation of the implicit scheme counteracts the energy gain of the explicit one. In fact such schemes are called symplectic. The basic idea is to go implicit on the velocity and explicit on the position. We have not found an elegant way to derive the scheme using complex numbers. In velocity-position space the equation for the next state is:

$$\begin{pmatrix} v^1 \\ x^1 \end{pmatrix} = \begin{pmatrix} 1 & -h \\ h & 1-h^2 \end{pmatrix} \begin{pmatrix} v^0 \\ x^0 \end{pmatrix}$$

The trajectories are now closed curves or curves that are bounded in phase space. Figure 6 shows several examples for different time steps. Interestingly for $h = 1$ we obtain a hexagon and for $h = \sqrt{2}$ we get a quadrilateral. For some cases such as $h = \frac{1}{2}$ the trajectory fills up a space bounded by two ellipses. Motivated by pure intellectual curiosity we have computed the time step that will produce any given n-gon. We achieved this by computing the eigenvectors of the matrix in the symplectic equation:

$$1 - \frac{h}{2} \pm ih\sqrt{4 - h^2/2}$$

We will not provide the details here. Note that from the eigenvalues we deduce that the method is unstable for time steps that are larger than 2 in this case.

The name of the integrator comes from the fact that the mapping preserves area, which is clearly the case for the above matrix since its determinant is equal to one. But why is it called "symplectic"? What does that word mean? An English dictionary defines it as: "Plaiting or joining together; - said of a bone next above the quadrate in the mandibular suspensorium of many fishes, which unites together the other bones of the suspensorium." Why name a mathematical property method after a fishbone? This is clarified in [11], the name was coined for other reasons by the famous physicist and mathematician Hermann Weyl. He had to name the property of a group he was working on and wanted to name it "complex." However that name was already taken to refer to an extension of the real numbers. So he replaced the Latin root "com" to its equivalent Greek root "sym" to concoct the word "symplectic."

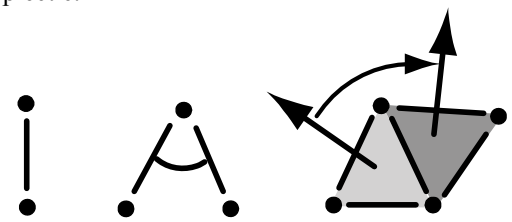

Figure 7: Three types of constraints: stretch, shear and bend.

5. Deformations as Constraints

The moral of the spring example is that it is a good idea to go implicit on the velocities and explicit on the positions once the velocities are computed. However, this procedure is still unstable for the case of springs. Therefore, instead of using springs we use hard constraints which can be softened if a bouncy behavior is desired. These hard links correspond to a resistance to stretch within a body. This is a relationship between two points of the simplicial complex, usually corresponding to its edges (1-simplices). We call this a type 1 constraint. Similarly we can define a type 2 "shear" constraint for each pair of 1-simplices by constraining the angle between them and we define a type 3 "bend" constraint between an edge connecting two 2-simplices. These three types of constraints are shown in Figure 7.

With these three constraints we are able to model the deformations of simplicial complexes of any dimension as shown in Figure 8. The number next to each simplex is the ratio:

$$\frac{\#\ \text{constraints}}{\#\ k-\text{simplices}}$$

The type of constraint has different interpretations depending on the k-simplices involved. For example, a type 2 constraint is a bending constraint for 1-simplices but a shear constraint for 2-simplices. Similarly a type 3 constraint is a twist constraint for 1-simplices and a bend constraint for 2-simplices. It is neat that we can model a wide range of effects using only three types of constraints.

We now provide exact mathematical expressions for these three types of constraints:

$$\begin{aligned} C_{1,(i,j)}(\mathbf{x}) &= \left|x_j - x_i\right| - l_{ij} \\ C_{2,(i,j,k)}(\mathbf{x}) &= \cos^{-1}\left(d_{ij} \cdot d_{ik}\right) - \gamma_{ijk} \\ C_{3,(i,j,k,l)}(\mathbf{x}) &= \cos^{-1}\left(n_{ijk} \cdot n_{jil}\right) - \theta_{ijkl}, \end{aligned}$$

where

$$d_{ij} = \frac{x_j - x_i}{|x_j - x_i|} \text{ and}$$
$$n_{ijk} = \frac{d_{ij} \times d_{ik}}{|d_{ij} \times d_{ik}|}.$$

For a given simplicial complex there will be many such constraints which have to be satisfied at the same time.

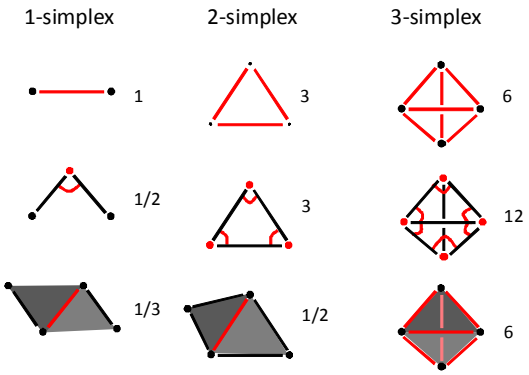

Figure 8: Three types of constraints for 1, 2 and 3-simplices.

Besides the three deformation constraints Nucleus also includes the following types of constraints (the list is growing and is not complete):

- Air model using drag and lift and a wind direction.
- Air pressure model for closed and non-closed meshes.
- Rigid Body constraint.
- Collisions (see Section 7)
- Point to surface constraint.
- Incompressible constraint for particles.

We can group all of these constraints in a vector of size m:

$$\mathbf{C}(\mathbf{x}) = 0.$$

This gives rise to a single non-linear system of equations for the change in velocity $\Delta\mathbf{v}$:

$$\mathbf{C}(\mathbf{x} + h\mathbf{v} + h\Delta\mathbf{v}) = 0.$$

Once the velocity change has been computed we can update the positions in an explicit manner:

$$\mathbf{v} = \mathbf{v} + \Delta\mathbf{v}.$$

$$\mathbf{x} = \mathbf{x} + h\mathbf{v}.$$

The big challenge is how to solve this highly nonlinear constraint equation. An idea we tried and was pursued independently in [10] is to linearize the equation as follows:

$$\mathbf{C}(\mathbf{x} + h\mathbf{v}) + \nabla\mathbf{C}(\mathbf{x} + h\mathbf{v})h\Delta\mathbf{v} = 0.$$

This results in a $3N \times m$ matrix equation:

$$Au = b.$$

The matrix is in general not square so a solution has to be found in the least squares sense. One such technique is to solve:

$$AA^T v = b$$

first and then to set

$$u = A^T v.$$

Alternatively one can use methods like LSQR or CGLS which require a black box routine that compute both the matrix multiply and its transpose. This technique works well as long as the constraints are close to linear which is true for small time steps. When the linear approximation is poor this procedure can actually make things worse by returning a solution that is far from the non-linear one resulting in instabilities.

Because of these problems we decided to solve the constraints in a sequential manner one at a time in a Gauss-Seidel manner [14]. For each constraint we do a line search along a direction $\mathbf{d}$ to satisfy the constraint:

$$f(\alpha) = c_k(\mathbf{x} + h\mathbf{v} + \alpha\mathbf{d}) = 0.$$

The search direction is chosen to be the gradient of the constraint. In practice this gradient can be quite tedious to compute analytically. There are two alternatives: one is to use automatic differentiation and the other one is to consider the direction orthogonal to all transformations that modify the constraint as proposed by Bridson et al. [6]. Once a direction has been chosen we can solve the above equation for the constraint using Newton iterations starting with:

$$\alpha = -\frac{f(0)}{\frac{df(0)}{d\alpha}}.$$

In fact we take only one Newton iteration per constraint since we have to satisfy many constraints simultaneously. Trying to satisfy one constraint accurately is pointless since other constraints might be

conflicted with it. In the next section we describe a way to deal with this problem.

6. Unified Solver: Resolving the Battle of the Constraints

To each type of constraint we assign an importance *I* and an order *O*. The importance lets Nucleus know how many times the constraint will try to solve itself. The order determines the sequence in which the constraints are being called. In Figure 9 we show an example of such a sequence. The evaluation is from top to bottom one row at a time.

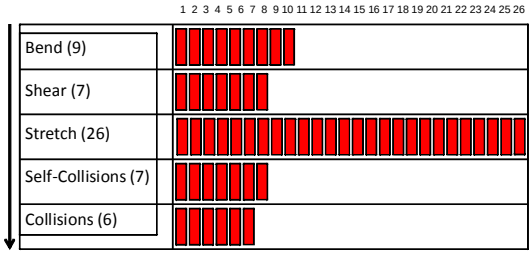

Figure 9: Order of evaluation and importance of several constraints.

The order of evaluation is important. We illustrate this with the example of an elastic band under tension between two bars as shown in Figure 10. If stretch is evaluated after the collisions then we get the results on the left. While if we evaluate collisions after stretch we get the situation depicted on the right. In general the latter is more desirable. However, notice how the band is overly stretched at the extremities near the bars. This is because we first attempt to solve for stretch which will shrink the entire band and then we resolve the collisions for the extremities. This is clearly not desirable.

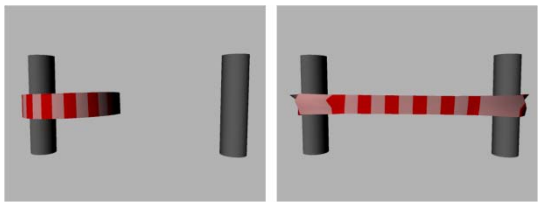

Figure 10: collisions followed by stretch (left), stretch followed by collisions (right).

To resolve this problem we interleave the constraint evaluations as shown in Figure 11. The solver computes from each constraint's importance the interleaving pattern and it also makes sure that all constraints are called in the final step. The math to do this is pretty straightforward and we will skip it in this paper. In Figure 11 the evaluation is done one column at a time from top to bottom as indicated by the arrow. In Figure 12 we show that this minimizes the order bias in the case of the rubber band under tension. There is no excessive stretching anymore at the extremities of the rubber band.

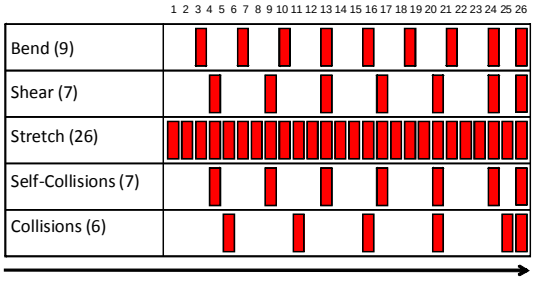

Figure 11: Interleaved evaluation of the constrained over one time step.

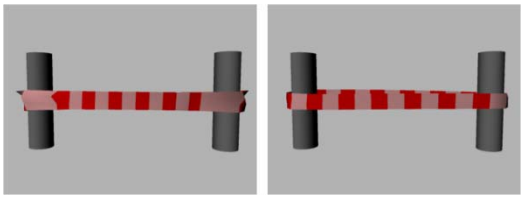

Figure 12: non-interleaved (left) versus interleaved (right).

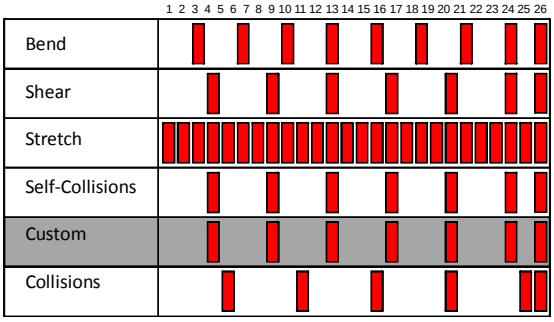

Figure 13: Custom constraints can be included in the Nucleus framework.

We have designed the solver such that users of the Nucleus library can potentially add their own constraints in the framework. The core of the Nucleus solver is blind to the internals of each constraint. Nucleus only knows about the importance and order of each constraint. For example, assume a user of the library wants to insert some code after each self-intersection call. In that case it can notify the Nucleus solver of the new constraint through an API and give it an order higher than self-intersection and same importance as shown in Figure 13. In the next Section we describe how we handle collisions in Nucleus. This is a very important component of the solver.

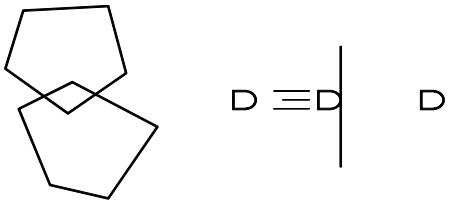

Figure 14: Penalty methods (left) versus space time collisions (right).

7. Collisions

We decided to present the collision handling after the general solver since it is quite involved and we didn't want to break the flow of the narrative of the basic methodology. Collisions differ from the other constraints as they are unilateral, which means that they are expressed as an inequality constraint:

$$\mathbf{C}(\mathbf{x}) \geq 0.$$

For example the constraint could be a function of the amount of overlap between two bodies at the end of the time step. However, in the case of fast moving objects it might happen that they do not overlap at the end of a time step. In practice if we do not want to restrict the size of the time step we have to take into account the entire space-time motion and detect collisions that might occur in between the frames. Figure 14 illustrates the difference: a fast moving bullet might be in a valid state at the end of the frame even though it collided with an object in between. Also for objects that are not closed: ones that do not unambiguously define whether a point is inside or outside, this approach does not work. Therefore we adopt a space-time approach in the Nucleus solver.

We will first explain collisions in a one-dimensional setting since we can conveniently depict the space-time picture in a plane. In fact solving the problem in one dimension is conceptually as hard as the three-dimensional case. The difference is just in the details of the computations involved.

In one dimension the particles are restricted to lie on a line. Consider the situation depicted in Figure 15, where two particles are approaching each other. Let the positions of the particles be denoted by a_0 and b_0 before the collision and by a_1 and b_1 at the end of the frame. Then a condition for the particles to have collided in between the time step is that:

$$V_0 \times V_1 \ < 0,$$

where

$$V_i = \ b_i - a_i \qquad i = 0,1.$$

In fact the time of collision can easily be computed in this case from these quantities as follows:

$$t = \frac{V_0}{V_0 - V_1}.$$

Once we have the time of collision we can resolve it as shown in Figure 16 either in an elastic manner (left) or in a completely inelastic manner (right) or some blend in between these two extremes. That is how we solve the one-dimensional problem

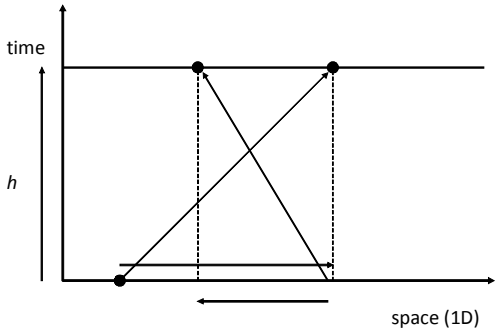

Figure 15: Space-time diagram of a one-dimensional collision.

In a two-dimensional plane two simplicial complexes collide through edge-point collisions only. Analogous to the one-dimensional case we compute the signed area of the triangle formed by the edge and the point at the start and at the end of the frame. If the sign of these areas is different then we know that the line defining the edge and the point have a collision somewhere in between as shown in Figure 17. However, this condition does not guarantee that the point actually hits the edge. This is only a necessary condition for a collision to occur. In this case we have to solve a quadratic equation in the time t to find the point of collision. Subsequently we move the point and edge to the time of collision and test whether the point is on the edge (similarly to Bridson et al. [6]).

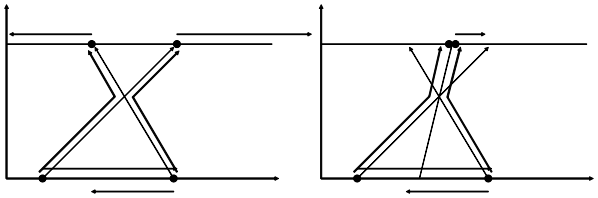

Figure 16: Elastic versus inelastic collision.

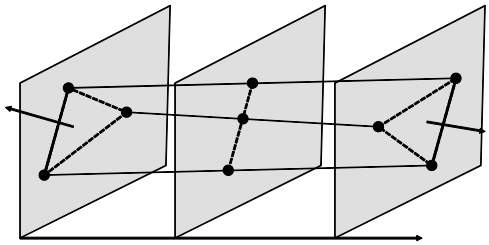

Figure 17: space-time collision of a point and an edge in two-dimensions.

In three-dimensions two simplicial complexes' simplices can only collide through point-triangle and edge-edge pairs. In both cases we can construct a tetrahedron formed by the pair as shown in Figure 18. These tetrahedra have a signed volume in three dimensions and similarly to the one-dimensional and two-dimensional cases we can check the signs of these volumes at the beginning and at the end of the time step. Of course we also check whether there is an actual intersection or not at the time of collision. In this case we have to solve a cubic polynomial equation to get the time of collision. In short this is the algorithm:

if $V_0 \times V_1 > 0$ stop
Find t such that $V_t = 0$
Check if primitives overlap at t
If yes handle collision.

If we add a thickness to each point, edge and triangle then the situation is a little more complicated and we have to consider the four cases listed in Figure 19. On the right we list the corresponding polynomial we have to solve to find the time of collision. This required us to write a routine that computes the real roots of a polynomial of degree up to 6. This is actually a lot trickier than it sounds, particularly due to numerical precision issues. How we dealt with these issues is beyond the scope of this paper.

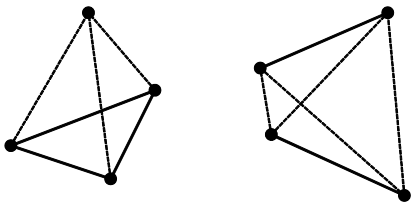

Figure 18: Both in the point/triangle and edge/edge case we can construct a tetrahedron.

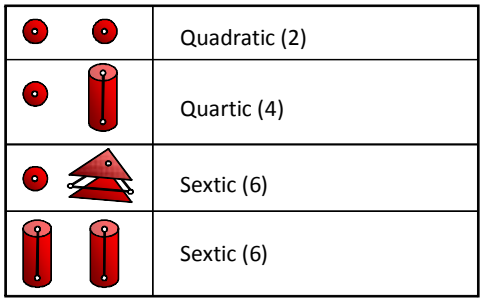

Figure 19: four possible collision types when thickness is added.

To handle many primitives we first use a hierarchical bounding volume structure to rapidly cull pairs that do not intersect. We then perform the more expensive collision tests described above on the remaining pairs. We have experimented with many different bounding volume data structures and found that a kDOP tree performed best in practice. We also use a hash table data structure for unstructured simplicial complexes such as a particle system. This is because the building of the topology (not its geometry) of the hierarchical tree can be quite costly in practice.

Many existing solvers resolve collisions sequentially as they occur in time and move the entire state to the time of collision. This is clearly the most accurate way to proceed. However, it can be computationally expensive in the case of many collision events. Worse it can suffer from lockups. For example, consider the case of a bouncing ball with a restitution coefficient smaller than one. It is a well known fact that the ball will bounce an infinite amount of times in a finite amount of time. Therefore an event based system would never halt in this case.

For these reasons we decided to adopt a fixed time step approach. We resolve the collisions sequentially but do not move the entire system to the time of collision. The sequence stops when all collisions are resolved or when a maximum number of iterations has been reached. Figure 20 shows a sequence of collisions for the case of three particles colliding in one-dimension.

We emphasize that our approach is an approximation. But perceptually it can be argued that it is hard to distinguish between a correct simulation and an approximate one in the case of many collisions. On the other hand our approach is more stable and does not suffer from lockup problems like event based approaches.

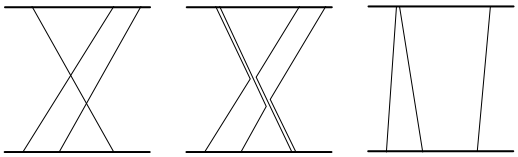

Figure 20: collision of three particles in one-dimension using a fixed time step approach.

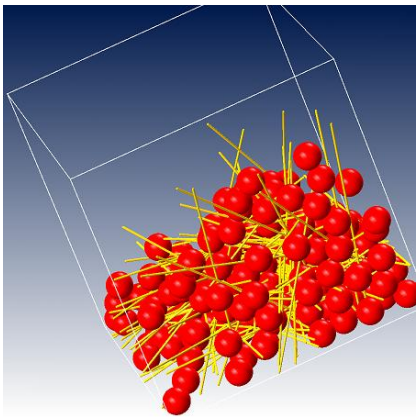

Figure 21: Snapshot of one of our demo programs.

8. Results

Nucleus was first released in Autodesk™ MAYA 8.5. It has been used in numerous productions by our customers. The capabilities of Nucleus are best demonstrated in animations and live demos. Most of the demos were derived from early prototypes but they use the same Nucleus library as MAYA does. Figure 21 shows a snapshot of one of our demos which runs in real-time and tests the various collisions between point, edges and triangles. Many demos and animations were shown by the author during his keynote talk at the IEEE CAD/Graphics International Conference 2009 in Yellow Mountain City in China.

Figure 22 shows a sequence of an animation of a Ballerina that we created with nCloth. The Ballerina is animated using key-frames and collides with the dress. The Ballerina's geometry changes quite a lot from frame to frame and our space-time approach turned out to be crucial to resolve the collisions. Figure 23 shows a sequence of a liquid simulation using nParticles.

My co-worker Duncan Brinsmead has a web blog with many interesting examples and unusual applications of Nucleus [7]. For example, he created a simulation of a slinky using nCloth. The shear constraint was particularly useful in this case to keep the slinky rigid as shown in Figure 24.

Figure 22: Animated sequence of a ballerina.

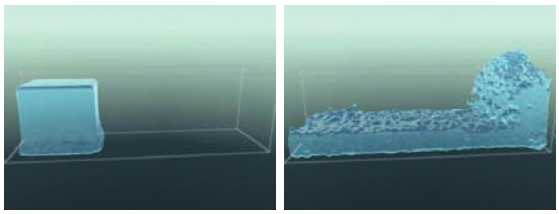

Figure 23: Simulation of a liquid using nParticles.

Figure 24: Still from an animation of a slinky using nCloth.

9. Conclusions and Future Work

In this paper we have given an overview of the new Nucleus solver library available in our Autodesk™ MAYA software. Currently only the nCloth and nParticle modules are exposed. However, the Nucleus solver also handles curve-like objects and other effects not yet exposed in Autodesk™ MAYA which were shown in our demos. In the future we intend to add more functionality to the solver such as true rigid bodies, better liquids, hair and other effects.

We are also exploring other uses of the Nucleus library outside of the field of computer graphics. Currently we are looking at applications in architecture [4]. We have recently obtained promising results in the area of panelization modeling which currently is a costly procedure.

Another very important area of future research is how to improve the control of Nucleus. Perhaps some controls can be built in as constraints. In fact we have one built in already which loosely constrains the cloth to a goal mesh animated by a user. Currently there are a lot of parameters in Nucleus and we would like to make the user-interface more user-friendly with more intuitive controls.

Also we are investigating ways to make the solver more multi-thread friendly. Some parts of the solver are already multi-threaded but the order dependent way in which we solve the constraints limits a global parallelization of the Nucleus solver.

We are currently working on all of the issues.

10. References

[1] P. Alexandrov, *Elementary Concepts of Topology*, Dover, New York, 1961.

[2] P. Alexandrov, *Combinatorial Topology*, Dover, New York, 1998.

[3] V. I. Arnold, *Mathematical Methods of Classical Mechanics*, Springer, New York, 1989.

[4] R. Attar, R. Aish, J. Stam, D. Brinsmead, A. Tessier, M. Glueck, A. Khan, "Physics-based generative design", CAAD Futures Conference, Montreal, 2009.

[5] D. Baraff, "Linear-time Dynamics Using Lagrange Multipliers", in Proc. SIGGRAPH 96, 1996, pp. 137-146.

[6] R. Bridson, R. Fedkiw, and J. Anderson, "Robust Treatment of Collisions, Contact and Friction for Cloth Animation", Transaction on Graphics (TOG), Vol. 21, No. 3, Proc. ACM SIGGRAPH 2002, 2002, pp. 594-603.

[7] D. Brinsmead, "AREA: Duncan's Corner", http://area.autodesk.com/index.php/blogs_duncan/tag_list/welcome/.

[8] R. Bridson, S. Marino, and R. Fedkiw, "Simulation of Clothing with Folds and Wrinkles", Proc. ACM/EUROGRAPHICS Symposium on Computer Animation, 2003, 2003, pp. 28-36.

[9] F. Faure, "Interactive Solid Animation Using Linearized Displacement Constraints", In Eurographics Workshop on Computer Animation and Simulation (EGCAS), 1998, pp. 61-72.

[10] R. Goldenthal, D. Harmon, R. Fattal, M. Bercovier, and E. Grinspun, "Efficient Simulation of Inextensible Cloth", Transactions on Graphics (TOG), Vol. 26, No. 3, Proc. ACM SIGGRAPH 2007, 2007, pp. 49-56.

[11] E. Hairer, C. Lubich, and G. Wanner, *Geometric Numerical Integration*, Springer, Berlin, 2000.

[12] T. Jakobsen, "Advanced Character Physics", Game Developer Conference, 2001.

[13] D. Kaufman, S. Sueda, D. James, and D. Pai, "Staggered Projections for Frictional Contact in Multibody Systems", ACM Transactions on Graphics (TOG), SIGGRAPH Asia 2008, Vol. 27, Num. 5, 2008, pp. 164-175.

[14]. J. J. Moreau, "On Unilateral Constraints, Friction and Plasticity", New Variational Techniques in Mathematical Physics, 1973, pp. 172-322.

[15] M. Müller, B. Heidelberger, M. Hennix, and J. Ratcliff, "Position Based Dynamics", Proceedings of VRIPhys'06, Madrid, Spain, November 6-7, 2006, pp. 71-80.

[16] X. Provot, "Deformation Constraints in a Mass-Spring Model to Describe Rigid Cloth Behavior", In Graphics Interface 95, 1995, pp. 147-154.

[17] D. Stewart, "Rigid Body Dynamics with Friction and Impact", SIAM Rev., 42, 1, 2000, pp. 3-39.

[18] D. Terzopoulos, J. Platt, A. Barr, and K. Fleischer, "Elastically Deformable Models", in Proc. SIGGRAPH 87, 1987, pp. 205-214