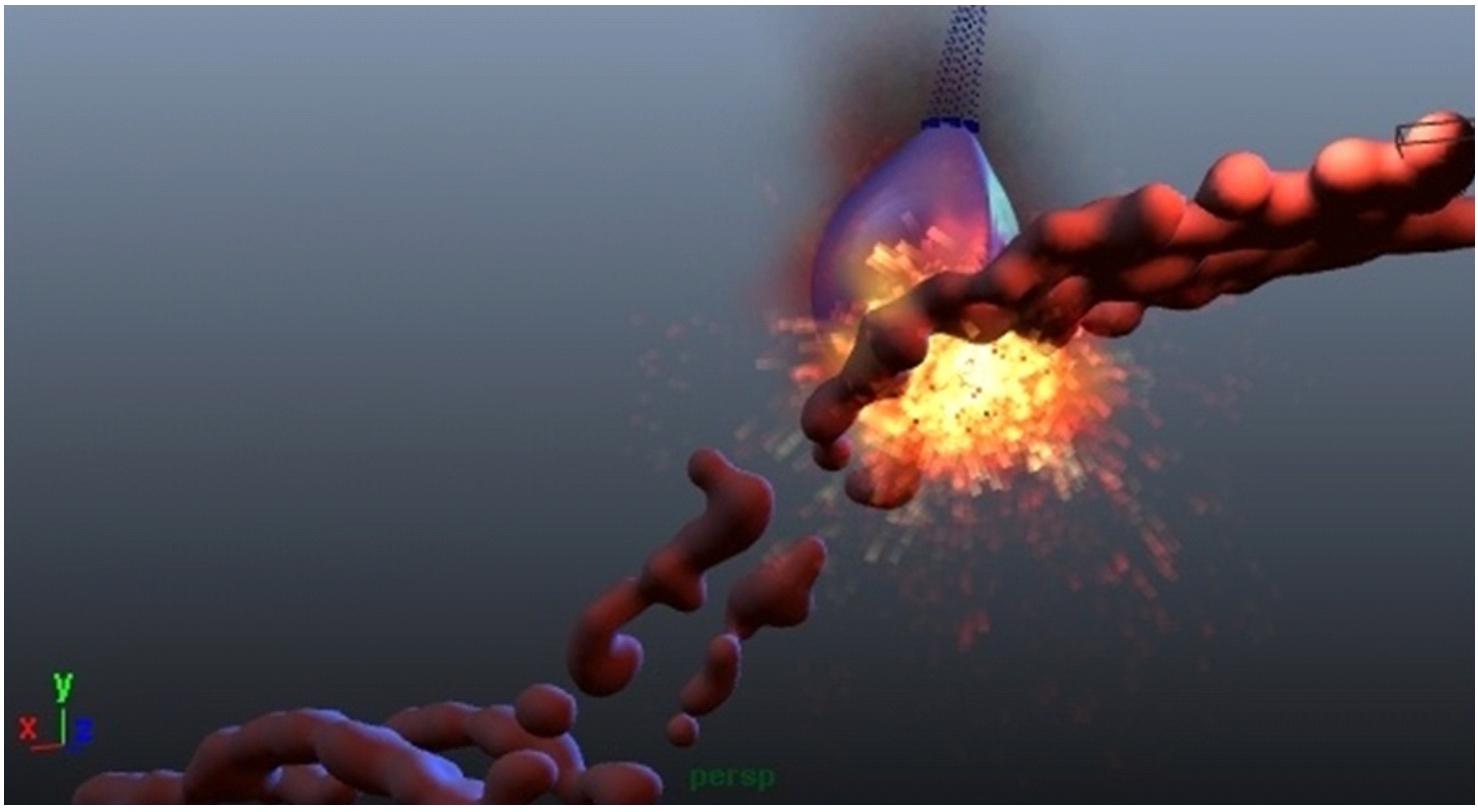
Autodesk®

Nucleus in Autodesk Maya



Nucleus in Autodesk Maya

Introduction

Nucleus was created to help address the need for a common Autodesk® Maya® software dynamics solver. By using a common solver, different dynamic effects can interact in complex ways that would likely not be possible with independent solvers. As well, the core solver is a separate component with no dependencies on the rest of Maya.

1.0 Introduction

1.1 What is Autodesk Maya Nucleus

Maya has several nodes (such as the Nucleus, Maya nCloth and Maya nParticle nodes) that help act as interfaces to the Nucleus solver, setting up relationships and translating data to and from Nucleus. This not only helps allow better interaction between different entities, but also takes advantage of stable solve and collision capabilities inherent in Nucleus.

This document discusses the internal solver architecture, and how the components of the solver are controlled through the Maya nodes and attributes. This is followed by a more detailed discussion on how to help achieve optimal results with Maya nCloth and nCloth caching, and some frequently used effects.

Much of this document will deal with the cloth application of Nucleus; however, nParticles can be considered a subset of nCloth in terms of Nucleus. Particles may be considered like cloth objects that have no bend or stretch resistance and vertex collisions and constraints only.

1.2 It's not just clothing

While nCloth was initially developed to help model clothing, there are many natural phenomena that can be created with nCloth, and now nParticles.

nCloth Effects

nCloth can be used for more than things like clothing and flags. It can also be used to help create rigid and semi-rigid objects that can break/tear, bend and deform. Additionally, there is a lift model for aerodynamic effects. Sample effects are:

- Clothing (with tearing)
- Flags
- Airplanes
- Balloons (inflating/deflating)
- Bridges
- Slinkys

nParticle Effects

nParticles can be used for typical particle effects like smoke and fireworks as well as more complicated liquid behaviors. Sample effects are:

- Smoke, fire
- Dissolving objects
- Pouring liquids
- Splashing
- Sticky fluids (mucus)

Nucleus constraints on nParticles

Nucleus constraints can be used with initial state nParticles to get the following effects:

- Liquid droplets rolling along complex surfaces
- Simplified rigid body behavior
- Suspended particles (tendrils)

1.3 Want help now? Read this first

If one is having problems with an nCloth setup and looking here for answers, first try the following:

1. Set Space Scale on the nucleus node based on one's unit settings. If one modeled in meters, set it to 1.0, but if one modeled in the default centimeters, set it 0.01. The main effect is to set gravity to the correct amount, as dynamics in Maya always interprets units as being meters regardless of the units setting.

2. For non-stretchy cloth, increase Stretch Resistance or apply a preset. One can find presets in the attribute editor for the cloth node.

3. Increase Substeps on the nucleus node to resolve collision problems or stretching in colliding regions.

4. Lower Lift on the nCloth relative to one's Space Scale setting. There is currently a problem where Lift has proportionately too great an effect at large scene scales. Thus, if one modeled in centimeters and set one's Space Scale to 0.01, then multiply the nCloth Lift value by 0.01.

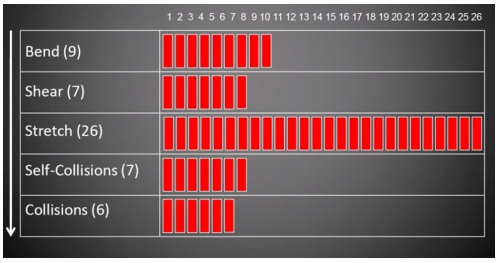
2.0 Core Solver Design

2.1 Building block design

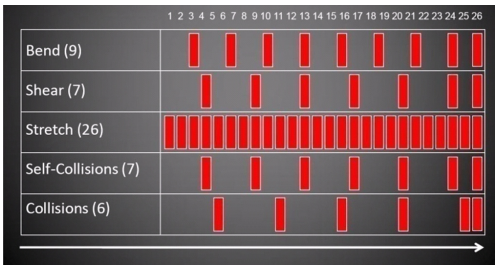
The nucleus solver has a multipurpose design where complex phenomena like cloth are handled by combining smaller sets of tasks. This building block approach helps allow the same solver to work for different types of simulations such as water, cloth or rigid bodies. The solve of these component parts is performed with implicit methods that help allow for stability for most setups. The general philosophy is to attempt to solve for large steps, rather than requiring small time steps. This helps allow the simulation to degrade relatively gracefully for faster low quality settings instead of locking up or becoming unstable.

2.2 Solving a multiply constrained system

A dynamic entity, such as cloth, typically requires that the solver compute many different effects and these effects may be in mutual conflict. For example, the vertices of a cloth draping over an object are affected by gravity, collisions with that object, stretch resistance of the cloth, and air friction. Gravity pulls the vertices down but the collisions, air friction, and stretch resistance all resist that downward motion. Each of these effects are a constraint on the vertices and must be iterated over time to solve the animation, although it may help to iterate more on some effects, such as the stretch resistance. Based on attribute settings the solver helps determine how many iterations to use for each effect in order to of simulation. The following tables show the way that the iterations for multiple effects are interleaved. The arrows represent the order of evaluation for both a sequential and an interleaved solve:

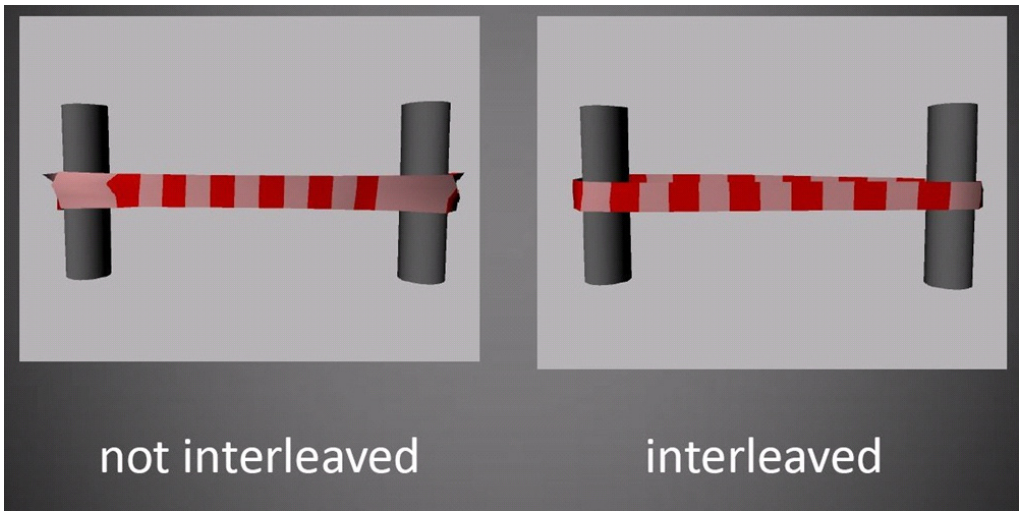


Non-Interleaved or Sequential Evaluation.



Interleaved (method used by Nucleus).

Collision evaluations are typically much more expensive to compute than stretch evaluations, so it is generally more efficient to have more stretch evaluations than collide ones, especially if the material is highly stretch resistant. However, if there are not enough collision evaluations interleaved with the stretch evaluations, one may see unacceptable stretching of cloth in regions of collision.



The above image shows how sufficient interleaving of stretch and collision results in less stretching along colliding faces.

The application of stretch and gravity help pull cloth The application of stretch and gravity help pull cloth into penetration with an object. When the collision iteration is applied, it helps push out the colliding regions, which causes stretch in the faces that are in collision. Further iterations of the stretch resistance will fix this stretch problem but bring the cloth back into collision. The last iteration in a step is always a collision evaluation, so without sufficient interleaving collision iterations among the stretch iterations, the effect will be that the last collide step helps to significantly stretch the cloth, regardless of how high the Stretch Resistance is. Increasing Substeps or Max Collision Iterations should deal with this problem. Extreme motions or accelerations will require more total iterations to solve.

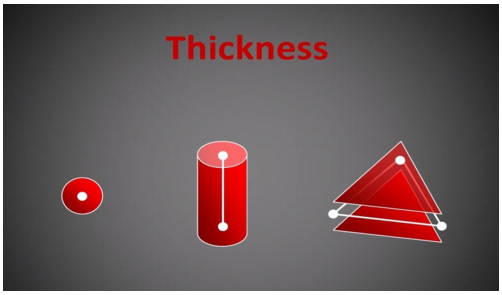
Internally, the solver works on simple entities such as stretch links between points. There is an initialization phase where the more complex items, such as constraint nodes and cloth, are broken down so the solver can more easily digest them. For example, when a cloth mesh is passed to the solver, a set of length constraints between point pairs is constructed. A constraint node may also build the same point pair lists. There are usually several lists built for effects such as collision and bend resistance. The numeric value of the corresponding attribute in Maya typically helps define the number of iterations of that effect over $1/24^{th}$ of a second, which by default corresponds to a single frame. Note that if Scaling Relation is not set to Per Link, then the iteration count per frame is also scaled by the number of polygons in the mesh. Per Link is the default Scaling Relation value, but nCloth attribute presets generally use the Object Space setting.

2.3 Hierarchical structure of Nucleus objects

The solver understands how to relate elements based on a simple hierarchy of dimension:

points (zero dimensions)
edges (one dimension)
faces (two dimensions)

(One can carry this further with volumetric objects like tetrahedra, although this is not currently used.)



The solver knows how to collide between these elements and will consider thickness, if set. Internally, full surface collisions (face to face) are handled by combining subsets of the simpler collisions: point to face and edge to edge. The collision computation may be thought of as computing the space-time intersection of the thickened components.

Also, each element can be constrained, or related to other elements. For example, one can constrain points to points, points to edges, and edges to faces.

2.4 Solver initialization and stepping

At the start frame, the solver helps construct a variety of structures, such as bounding trees, to help allow for efficient computation. Also, the initial state of the input object at the start frame is cached, which helps allow the solver (along with other effects) to keep interpenetrations from exploding the mesh apart. At the start of the simulation, the constraint one defines is broken down into lists of links and low level relationships that can be solved more quickly. For Slide on Surface constraints, the initialization involves computing nearest point on surface along a normal, which is a relatively expensive computation. Note that with constraints, if one keyframes the Enable attribute on the dynamicConstraint node, the constraint will be reinitialized on frames where the enable turns on. This is useful for having a hand grab a dynamic object.

Maya animation (i.e. keyframing and deformation of objects) is computed once per frame (or step) and the internal nucleus structures that the solver works on are updated from this animation. This step is further broken down by the Nucleus system into substeps, which are smaller divisions of time within the frame. Animation curves and deformers are not evaluated during these substeps, although their effect is interpolated in a linear fashion within the substeps.

Even if there is only one substep, passive objects have a velocity per vertex that captures the change in shape across the frame which is used for the collisions (one may think of the surface extruded in time as a 4 dimensional object). Note that in some cases, simple linear interpolation of animation across a step might not be sufficient. To help handle this, one can set the value for Evaluate every X frames, in the Cache Options, to make the base step size finer.

Within a substep, there may be finer iterations of effects like stretch resistance and collisions, depending on the settings. These finer iterations are performed across the time step defined by the substep. For example, if there were 10 collision iterations within a substep, each of those iterations would perform the collision computation for the delta time of the substep, rather than stepping forward in time by smaller increments. Thus, the substep represents the smallest unit of time used by the solver. Within a substep, one can have more iterations, but each of these iterations attempts to solve across the time interval defined by the substep. Values like Stretch Resistance and Bend Resistance are defined relative to time, not substeps, although changing the substeps can affect the quality of the solution for these attributes, which changes the effect. However, the solver attempts to preserve the defined level of stretchiness, for example, in a manner independent of substeps. By default, if one sets Stretch Resistance to 10, there are 10 iterations of stretch resistance per frame. If Substeps is set to 2, each substep has 5 iterations of stretch resistance. If the Substeps is increased to 10, there is 1 iteration of stretch resistance for each substep. However, if the Substeps is increased to 20, there is now 1 iteration of stretch resistance per substep, or 20 stretch resistance iterations per frame. The intensity of the stretch resistance evaluation is reduced per step in such a way that the total stretch resistance stays roughly the same, even though there are twice as many evaluations.

Note that, in addition to effects such as stretch resistance, bend resistance, and rigidity, collide and self collide iterations are also defined relative to time. Thus, if the collide iterations (Max Collision Iterations) were set to 10 and the Substeps to 2, there are 5 collide iterations per substep. In some cases, increasing substeps can actually help make the simulation solve faster. This is because the number of total collision iterations might stay the same but each iteration occurs over a smaller timestep, which is more easily computed due to less collision pairs per step. (Collision pairs is the list of elements which have overlapping boundaries in space-time. For self collisions, this list gets large when the thickness is greater than the face size, especially with full surface self collision, which increases computation time.) Substeps carry the computation price of recomputing the internal bounding tree of all objects, which is expensive. This results in the situation where it helps to have collide iterations somewhat higher than Substeps for maximum efficiency.

Another useful rule of thumb relates the collision iterations and stretch. In general (assuming that the object's Scaling Relation is at the default setting of Link), the collision iterations should be at least 1/10th of the Stretch Resistance value, so that at least one collision evaluation occurs every 10 stretch resistance evaluations.

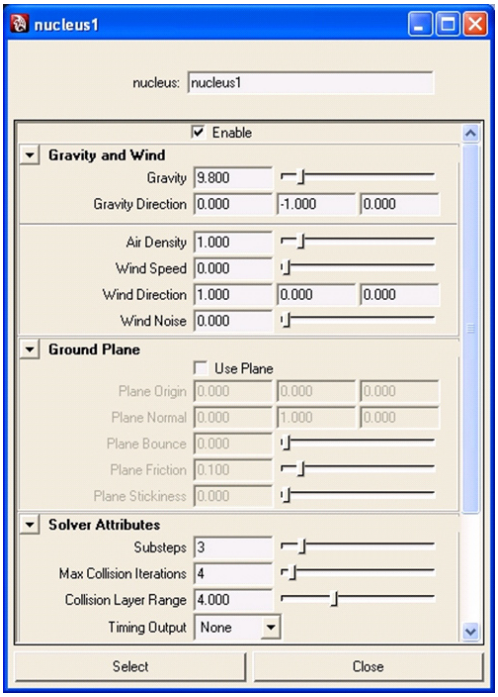
When simulating nParticles with minimal constraints, meaning just collision with objects and no self collisions, a Substep value of 1.0 and Max Collision Iterations of 1 may be sufficient. The default Substeps value is 3, so, for simple particle simulations, it often helps to lower the Max Collision Iterations and Substeps. Higher iterations are generally required when one has constraints that need to propagate across a network, for example multiple elements in simultaneous collision like stacked balls or the set of interconnected links that handle stretch resistance on a mesh. In general, the iteration requirement goes up exponentially with the complexity of such networks.

3.0 Maya Dependency Graph Structure of Nucleus Objects

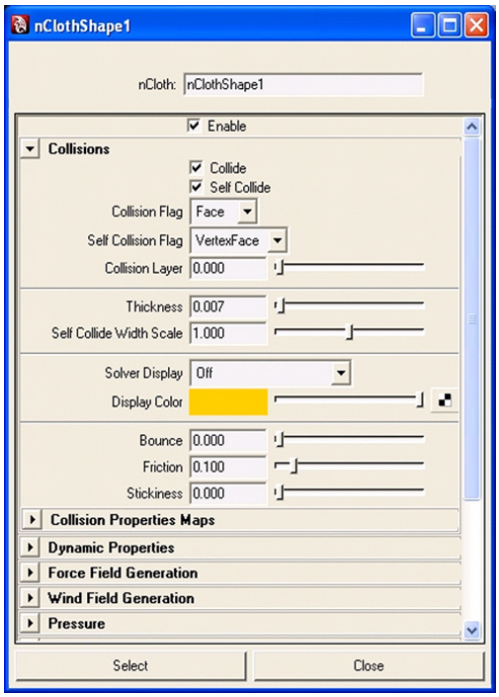
3.1 Node types and inheritance

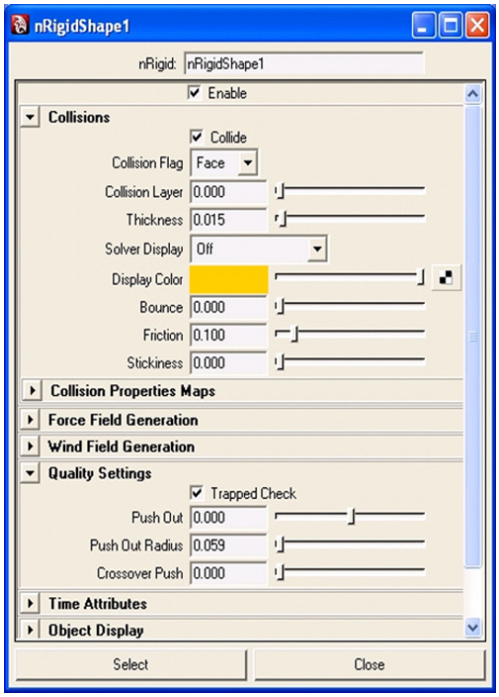
There are basically four classes of nucleus nodes in Maya: the Nucleus solver node, object node, dynamic constraint node, and the Nucleus component node.

3.1.1 Nucleus solver node



3.1.2 Nucleus object nodes



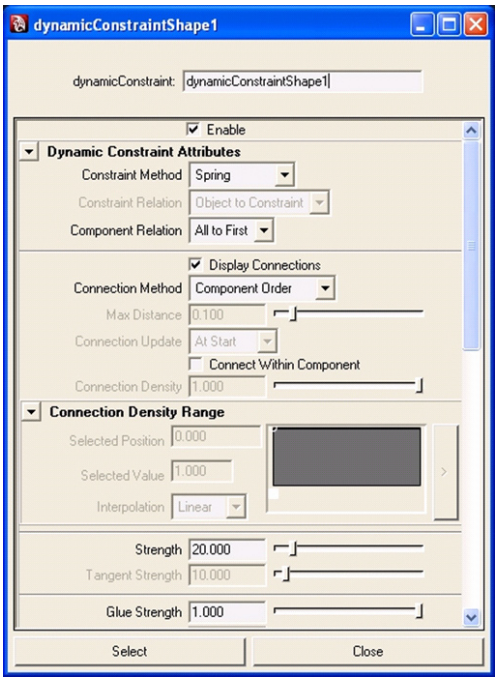


Currently, there is only one flavor of solver, and its node type is nucleus. The nucleus node connects to the entities involved in the solve, such as nCloth and nParticle objects, and contains attributes that affect of the evaluation of its connected entities. The nucleus node can be thought of as a contained world or universe. Objects can only interact if they are connected to the same nucleus node. The nucleus node also contains global values, such as wind and gravity, which affect the objects that are connected to it.

Wind and Gravity can be ignored on an individual object, and similar to Maya Fields, they can be applied to individual Nucleus objects. However, the wind model in the Nucleus solver knows about surface orientation, which Maya wind fields do not. This helps allow cloth to have more air drag when facing the wind than edge on, which is important for realistic cloth and wind interaction. Also, Nucleus Wind and Gravity are computed within substeps, while external fields are computed only once per frame. Thus, the Nucleus Wind may better integrate with effects like stretch than a wind field would. (This situation may change in future with customization and extension fields for nucleus.) The Nucleus node is the one that helps invoke the solver for each new step when the time is changed.

Nucleus objects are the entities that are operated on by Nucleus. Currently, there are nParticle (nParticleShape), nCloth (nClothShape), and nRigid (nRigidShape) nodes. The nRigid node currently helps define passively colliding objects only, but may at some point have an active toggle if rigid body simulations are added to Nucleus. The nCloth, nRigid, and nParticle nodes inherit attributes and behaviors from a shared base class, which in turn inherits from Maya classic particles. This helps allow Maya to share existing functionality where appropriate, such as classic particle emitters, expressions, events, rendering for nParticles, and Maya fields for nCloth and nParticles. Currently, this structure does result in nodes having some inherited attributes that are not used. These attributes are suppressed in the Maya user interface (UI), such as the Attribute Editor, but they may still be visible in other places such as the Channel Box. Some inherited functionality, such as particle expressions on nCloth, is not yet fully supported.

3.1.3 Constraint nodes



Constraint nodes help define relationships between Nucleus objects, or between the components of Nucleus objects. Currently, the manner of constraining is handled through a single, multipurpose node called dynamicConstraint. At the simulation start frame, this node helps build low level connections and relationships used by Nucleus during its solve. The constraint node also builds connections on frames where its enable flag (Enable attribute) is keyframed on. Keyframing the Enable attribute helps allow one to handle effects like animated grabs. For example, one can parent a Transform constraint for some nCloth vertices to a hand. As the hand grabs the cloth, the constraint can be enabled, which helps establish the rest position. As the hand continues to move, it pulls the cloth with it. Note that for some constraints, such as Slide on Surface constraints, the construction of the links may be time consuming, which could result in a slow rewind to the start frame or file load.

The constraint has input nComponent nodes and it helps define how to relate these nComponents. The Component Relation attribute helps define how to relate the connected components: All to All, All to First, and Chain (Chain is a daisy chain based on the order the components are connected into the constraint.). For example, if one had several nCloth meshes that one wished to constrain to a body mesh, it would be possible to constrain them with a single constraint where body component was the first one connected to the constraint (connected to constraint. componentIds[o]), and the Component Relation was set to All to First. If one wanted to form a chain of connections between a set of objects, one could use Chain as the Component Relation, although note that this relies on the order in which nComponents are connected to the constraint componentId array. Note that it is also possible to have multiple nComponents for the same nCloth (useful for specific lists of self links), although this cannot be setup through the UI currently.

The toggle, Connect Within Component, helps define if links are made within nComponents, not just between nComponents. For example, if one wished to stiffen a shirt collar, one could have a single nComponent that specified the collar vertices, and on the constraint, select Connect Within Component so that the vertices will interconnect.

The nComponents may have different types, and the constraint attempts to make the most logical connection based on the type being connected. For example, a point being constrained to an object will connect to the nearest point on the object, and the link will have stiffness relative to the orientation of the surface, not just the point on surface. Likewise, links to faces are affected by rotation of the face, while point to point links are not affected by surface orientation (if there even is a surface). A link between two faces will be stiff, while a link between two edges is freer to rotate like a hinge. Also, links between edges connect to the nearest point on each edge. Thus, if one connects the border edges of two planes that are just touching, the planes will stay attached along that border as they animate. If one instead connected the border vertices and the vertices did not exactly lineup, the planes could separate a bit.

The dynamicConstraint node is most typically used for spring style links between components; however, it can also do things like collide between the components, apply a motion drag effect, apply a force, or disable collisions between components. The links between entities can also be dynamically broken by using the Glue Strength attribute.

The nConstraint menu, which is used for creating constraints, has different items that are basically presets for common constraint setups (for example, Transform, and Point to Surface). However, there is a much larger set of effects that are possible by editing attributes on the dynamicConstraint and nComponent nodes. There are also some setups that may be useful which require custom node connections. It is currently not possible in the UI to create two nComponents nodes for the same nObject that feed into the same constraint. However, this is a useful setup if one wants to create lists of explicitly linked vertices, and by scripting node setups it is possible.

Note that constraints on nParticles currently work only with initial state particles, because the constraint builds the links at the start frame. However, one can still use the older style springs for emitted particles.

3.1.4 nComponents

The nComponent node helps define the list of elements that a nObject passes to a constraint. This list may include vertices, edges, triangles, or the full object, and it is automatically constructed when one creates a constraint through the Maya UI. One may specify elements through an explicit list (array) on the nComponent, or the set of elements may also be implicitly defined as all elements or only elements of a particular type that lie along a border. Implicitly defined components have the advantage of continuing to work even when one changes the topology of the mesh. For example, one can define a constraint between two cloth planes such that boundary edges that are near to each other will be linked. One can then change the resolution of one of the cloth meshes, and the constraint will continue to work. If the connections were explicitly defined using edge indices, the connections might be between the wrong edges after a change of topology.

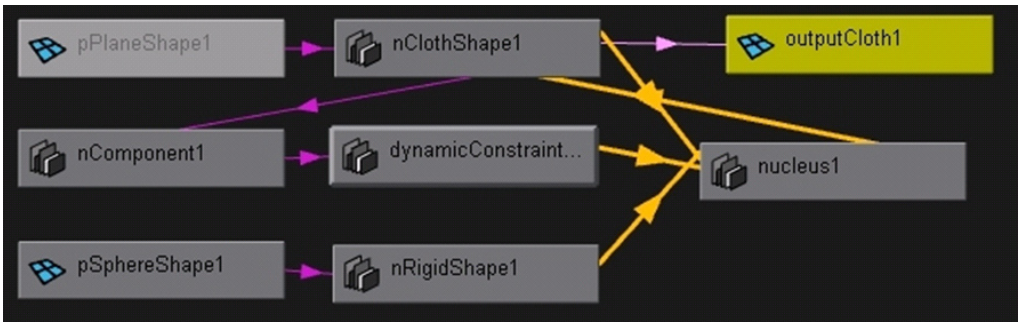
Note that the nComponent must be all of a particular type:points, edges, faces or object, although, different nComponents for the same constraint may have different types. Also note that for nParticles, the only valid nComponent type is points.

In addition to defining parts of an nObject to constrain, the nComponent has attributes for controlling the relative weighting of the component for the constraint. If one sets the weight of an nComponent to 0, it is affected by other components but does not affect other components: it is effectively a 1-way constraint. By default, all constraints set the component weights to 1.0, and thus are two way constraints, although it should be noted that all links to nRigid nodes (i.e. passive colliders) are effectively 1-way, because the nRigid cannot be moved by the simulation.

nComponents can connect to Maya sets instead of using the explicit array on the nComponent. This is set up when one creates a constraint with the Use Sets option turned on. However, it adds a lot of nodes and connections, and does not really provide much in the way of making the constraint adapt to changes in nObject topology. Thus, you should avoid using sets with Nucleus constraints unless you are aware of this situation and are prepared to deal with it.

3.2 Dependency graph connections

Nucleus entities connect to the nucleus node with two connections: one that supplies start frame data and one that supplies current frame data. When the nucleus node detects that time has changed to the start frame, it pulls on its input start state connections, causing each of those nodes to compute and set the start data on its internal nucleus structure. A runtime Id is passed through these connections instead of the actual Nucleus data. The connections serve two purposes:



Simplified view of dependency graph connections for a cloth plane with a transform constraint along with a colliding sphere (nRigid).

they help define the structural connections for the internal nucleus solver and they help trigger the setting of internal solver data from the dependency nodes. In the case of nCloth and nParticles nodes, there is also a connection from the nucleus node back to them, for purposes of updating nCloth and nParticle output data from the internal solved data.

Above is a simplified view of dependency graph connections for a cloth plane with a transform constraint along with a colliding sphere (nRigid).

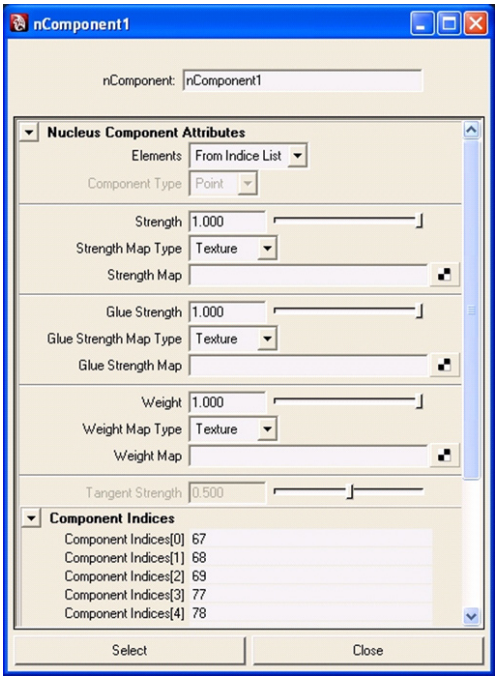
The different nObject nodes (nClothShape1, dynamicConstraintShape1, and nRigidShape1) connect to the nucleus node. The nCloth is the only node that requires a return connection from the nucleus node because it needs the output of the solver in order to set its output position. The outputCloth1 node is the final output of the entire simulation.

The general flow of data through the dependency graph is as follows:

1. The output value of an nObject is requested (for example an nCloth output mesh, or an nParticle draw).

2. The nObject helps evaluate its connection to the nucleus node outputObjects array.

3. The nucleus node helps check to see if its current state is up to date. If not, it helps evaluate all its inputs needed for the current solve: either the start inputs if at the start frame, or the current inputs if at an advanced frame. This will trigger current or start frame evaluations on its input nodes. Once the inputs have been evaluated, it triggers the internal nucleus solver for that frame and returns the requested value back to the nObject. This tells the nObject that the internal solver state is now up-to-date, and the nObject then sets its output from the internal solver state.



When nObjects gets a pull on their output start state connections, they do the initialization of the internal nucleus objects. For nCloth, this means building a triangulated mesh for collisions and also defining arrays of links that handle stretch and bend resistance. When nObjects get a current frame evaluation, they generally only set internal attributes values from the nodes, meaning data structures that may change over time: the input attract mesh and evaluate Maya fields.

Dynamic Constraints nodes have input connections from nComponents. There is generally one nComponent for each nObject that participates in the nConstraint (although potentially one could setup multiple nComponents for the same nObject).

Autodesk®

4.0 Nucleus Quality Settings

The primary quality setting in Nucleus is the Substeps attribute on the nucleus node. The more substeps one has, the more accurate the simulation. However, increasing the Substeps is relatively compute intensive and may not solve certain problematic cases, such as when a simulation starts out in a bad state. For these cases, Nucleus also has other less general purpose quality attributes such as Pushout and Trapped Check.

4.1 Substeps

The default Substep setting of 3 is generally not high enough for detailed real world cloth models. Therefore, the first thing to try when encountering collision failure or excessive stretching is to increase the Substeps. Nucleus helps achieve better performance by a sparse interleaving of iterations of various computations like stretch and collision. Each computation performs at least one iteration per Substep. As one increases the Substeps, the interleaving will become less sparse and effects will better interleave with each other. For example, if the Max Collision Iterations is 4 and the Stretch Resistance is 20, there will be one collision iteration for every 5 stretch iterations as long as the Substeps is 4 or less. If the Substeps were raised to 10, there would now be 10 collide iterations for the frame, and thus 1 for every 2 stretch iterations. If Substeps were raised to 30, there would be 30 stretch and collide iterations per frame, and thus a 1 to 1 ratio. In general, stretching is not too bad when there is at least 1 collide iteration for every 10 stretch iterations. So, for a Stretch Resistance value of 500, one might want the Substeps at 50.

Substeps also breaks up time into smaller chunks, which is especially helpful when objects are moving quickly, or when there are many close inter-colliding elements. There are even cases where increasing Substeps can speed up the solve, although in general, it will slow things down because some values may have more iterations per frame and the bounding structures are recomputed for each substep.

4.2 Max Collision Iterations

Max Collision Iterations help determine the number of base iterations for collision within a step (or frame). It is perhaps poorly named, because the total collision iterations for a frame will be higher if the Substep value is greater than the Max Collision Iteration setting. One may think of the collision iterations as being at least as high as the Substeps. Making collision iterations lower than Substeps has no effect. In general, it can be useful to have Max Collision Iterations somewhat higher than the Substep value, because it is generally faster to compute a collide iteration than a full Substep. A Max Collision Iterations value around 1.5 times the Substep value can work well, although one is encouraged to experiment to find a value that works well for their setup.

Note that internally, Nucleus may also compute 5 sub-collision steps. These sub-collisions only kick in when a given collision iteration still has elements in collision after a given iteration. One cannot currently set these internal sub-collisions.

4.3 Collision Flag and internal tessellation

The Collision Flag attribute helps determine if vertices, edges or faces of a mesh will collide. The radius of collision about these objects is determined by the Thickness attribute, or the Radius attribute in the case of nParticles. Note that for face collisions (Collision Flag set to Face), the solver will internally triangulate faces with more than 3 vertices. For quads, this amounts to a left "quad flip", and the quad flip of the output nCloth mesh is automatically set to left so it will help match the internal tessellation used by Nucleus. For most nCloth, using the Face Collision Flag is generally the most useful and it is relatively fast. Note that larger Thickness attribute values can help allow for lower Substeps and Max Collision Iterations settings, although care needs to be taken if one also has self collisions (See Self Collision quality settings).

4.4 Pushout and Trapped Check

Both Trapped Check and Pushout make use of triangle normals to define inside from outside of an object, and then they help push vertices outside an object. Trapped Check is the more efficient version of this process, and is only applied to vertices that are currently interpenetrating. Trapped Check only works with collisions enabled (the Collide attribute turned on), while Pushout can be applied even if collisions are off.

Note that Self Trapped Check does not help push along normals. Rather, it simply helps disable self collision for vertices that belong to self interpenetrating triangles.

Pushout always has an effect, even if Trapped Check and Collide are turned off. (One can actually use Pushout instead of collisions.). If a mesh has Pushout on it, then vertices of other nObjects will be helped pushed along the normal, if they are within the Pushout Radius. Pushout tends to get slower for large Pushout Radius values. Pushout works along the normal, so it should be avoided if one wants to collide both sides of a surface. When the Pushout is set to 1.0, it tries, in one step, to move the vertex out of interpenetration (including thickness values). However, in this single step, other effects like stretch may tend to pull the vertex back into collision (Increasing Substeps will help avoid this.). Note that Pushout values greater than 1.0 might result in an effect like an egg on a frying pan, where vertices are pushed up off a surface then continually fall back into the surface instead of being pushed up just to the point of collision

The Pushout Radius should not be confused with notions of collision envelopes or thickness. Nucleus has a well defined notion of thickness for collisions that is separate from the pushout. The pushout may be thought of more as an applied force that goes out to the distance of the Pushout Radius from the surface (on the side opposite the normal). Pushout helps make most sense for closed objects that have an interior. Points on the inside of the interior are repelled to the outside of the object. When the Pushout is 1.0, then for a given iteration, it pushes points outward just to the point where the surfaces just touch. Note that the pushout is always points being pushed out of the mesh, not triangles, so there may still be interpenetration along triangle edges after pushout.

Trapped Check is like Pushout, but it only works when collisions are on (the Collide attribute turned on), and just for triangles that are in collision. So, if at the start frame, one has two objects that are intersecting, Trapped Check helps push out the vertices of intersecting faces. This is more efficient than using Pushout, if one already has collisions on, but it will not push out triangles that are not interpenetrating. For example, one could put a cloth sphere fully inside a cube and use Pushout on the cube to help push the sphere out of the interior of the cube. Trapped Check cannot do this. Trapped Check is turned on by default for passive collision objects (which are assumed to be closed surfaces like a body), while it is turned off by default for nCloth objects, which tend to be two sided sheets.

Note that there was a bug in versions prior to Maya 2008 where Trapped Check was computed even when collisions were turned off.

4.5 Self Collision quality settings

After Substeps, the Self Collision Flag and Thickness are the most critical attributes for good self collision quality. Max Self Collision Iterations may be useful, although one can usually manage just by increasing Substeps. Also, Self Trapped Check can help with stuck vertices. Self Crossover Push is generally less useful, and should only be used as a last resort. In general, Nucleus currently does not have the best techniques for resolving meshes that are initially modeled in self collision, thus it is important that the model be initially free of self collision. Collision between different objects at the start frame can be resolved using Pushout and Trapped Check, while Self Trapped Check and Self Crossover Push do not work nearly as well (This is because self collisions cannot be disambiguated by simply looking at a normal.). Note that the relative thickness between colliding components is adjusted to be never greater than their separation at the start frame. This helps keep the mesh from blowing up when the self collision thickness is larger than the triangle size,

and additionally, it helps keep objects from blowing up where they are in collision or self collision at the start frame.

4.5.1 Self Collision Flag and Self Collide Width Scale

For self collisions, it is important to understand the relationship between collision thickness and the method used for self collision. Thick self collisions can be slow to compute, especially when the Self Collision Flag is set to Full Surface.

The Self Collision Flag initially defaults to VertexFace for a simple mesh and to Vertex for a complex mesh. The nCloth creation attempts to initialize the thickness and self collision thickness to good values based on the average triangle size. For a dense mesh with the vertex self collision, the Self Collide Width scale is set to 3 so that the thickened vertices roughly touch. (One can see this by setting the Solver Display to Self Collision Thickness.). If one changes the scale or resolution of an nCloth object after creating it, these default settings may no longer work well.

Self collision performance is largely determined by how many collision pairs are generated. For thicker collisions, one will get more pairs. This is significant because processing collision pairs is one of the most computationally intensive tasks. A flat mesh may have no self collision pairs if it is thin, but as one increases the thickness, vertices may begin to collide with edges on opposing triangle neighbors, thus performance at a certain thickness could become dramatically slower. This is why thickness is more of a performance issue with self collisions than for collisions between objects. If one wants thick collisions between objects and needs full surface self collision, then lower the Self Collide Width Scale to help keep the solve from getting too slow.

Vertex-to-vertex collisions are much faster to compute than other types, so, for a dense mesh with uniform sized triangles, it often makes sense to use vertex self collisions only. In these cases, it is frequently useful to increase the Self Collide Width Scale to the point where vertices just overlap with no holes to fall through. This case often may solve much faster than thinner full surface collisions. Also, due to the thickness of the self collision, one may get away in some cases with lower Substeps or Max Self Collide Iterations values. Another advantage of vertex collisions is that the self collision thickness helps provide a type of natural bend stiffness that can work better than using the Bend Resistance attribute. However, if Substeps and/or Stretch Resistance iterations are low, holes may open up where surface stretching allows vertices to push through into self collision. Thus, having a baseline for Substeps and Stretch Resistance iterations values may be important for vertex style self collisions.

One may still need to use the Full Surface collision flag (Self Collision Flag set to Full Surface), particularly when there are large faces in the mesh or when one needs self collisions that are thinner than the triangle size. For closed objects, such as a cloth sphere, Vertex Face collisions will be faster than Full Surface, and they should be just as good. However, in cases where edges can self collide such as in the region where a shirt buttons at the front, Vertex Face may have problems, and Full Surface would be required.

4.5.2 Max Self Collide Iterations

Max Self Collide Iterations helps determine the base iterating per step for self collisions. In general, one can set this attribute to the same value as Max Collision Iterations (on the nucleus node), or even ignore it and rely on Substeps because Max Self Collide Iterations will never be less than the Substeps value. For efficiency, one could key it to a higher value just for specific problem area or for fast moving frames.

4.5.3 Self Trapped Check

Unlike Trapped Check, Self Trapped Check does not look at normals and do a push. Rather, it simply helps disable self collisions for vertices that belong to self interpenetrating triangles. This can be useful in areas where pinching occurs, such as when a character's elbow interpenetrates with its belly, causing self collision failure. Self Trapped Check can help keep vertices from getting stuck in such regions. However, there is a little added computational overhead to using Self Trapped Check. In some cases, it can cause self collision failure to get worse. One way of using Self Trapped Check might be to key it on just for problem frames, and preferably, only when the cloth is pulling apart.

4.5.4 Self Crossover Push

Self Crossover Push can, in some cases, help untangle places where, either self collisions have failed, or the surface was in self collision at the start frame. It does not use the surface normal in the same manner as Pushout, but rather it helps push along a tangent direction for overlapping faces. It can be slow, and in some cases, may make self collisions worse. It is best to try Self Crossover Push only when other methods fail.

5.0 Matching Real World Conditions

Many of the attributes in Nucleus: Friction, Stretch Resistance, and Stickiness, do not correspond directly to physical coefficients, and thus must be set by testing and matching observed behaviors (if one has a real world reference to match). However, there are some basic elements one can set based on real world units.

5.1 Adjusting Gravity

Gravity is the most important basic attribute, and it strongly affects overall speed and sense of scale. In Maya, Dynamics and nDynamics do not adjust to the current unit settings in the preferences, and the default gravity treats a unit as being a meter. If one models at the default units setting of centimeters, the gravity will be off by a factor of 100, and objects will fall as if they are enormous or in slow motion. Note how scale and time are related here—slow down a guy in a lizard suit and one has Godzilla. If one's units are centimeters, then set the Space Scale on the nucleus node to 0.01. The primary effect of this adjustment is to make gravity 100 times stronger. Also note that, if gravity is 100 times stronger, the simulation needs to iterate 100 times more per frame to help achieve the same quality, in terms of stretch resistance and other similar effects. (This assumes that gravity is the sole driver of motion in the scene).

Another way of looking at this scenario is that the simulation now moves 100 times as fast, and to help accurately model this change, one needs proportionately more simulation steps. In general, one needs to increase many attributes to preserve the same behavior when increasing gravity, in particular Stretch Resistance, Bend Resistance, Compression Resistance, and Substeps.

Note, however, that gravity is usually not the sole driver of motion. If one has cloth on a moving character, the quality generally needs to be high enough to handle the maximum acceleration of the body. For something like a kick boxer, the accelerations can be much greater than the effect of gravity.

5.2 Mass

Mass is important for two properties: interaction of objects with each other and interaction of objects with air. When objects collide with each other, they transfer momentum based on the relative mass. Note that Mass does not affect the acceleration from gravity, although it does affect drag from air. The drag may be thought of as collision of the object with air molecules, and the relative mass of the air is part of the Air Density attribute on the nucleus node. In terms of the physics, the mass does not need to be in real world units for proper collisions as long as the relative values are the same (For example, if object B is twice as heavy per unit volume as object A, that makes the mass of B twice that of A).

However, there are two factors that help make Mass determination more complex: air resistance and point density. The collision interaction is handled using a point mass, where the surface area of a cloth or the radius of an nParticle is ignored. If, for example, a very dense cloth mesh collides with a very sparse mesh (i.e. large triangle size), it will

seem proportionately more massive. In a similar fashion, if one makes particle water with many small particles then collides it with another system that has fewer large particles, the smaller or denser water will seem to be heavier, even though both systems have the same Mass value. Therefore, one may want to adjust the Mass to reflect the size of particle or the general area of coverage of a cloth vertex.

5.3 Drag and Lift

Currently Drag needs to be factored in based on observation. Even if one knew the physical air density (air mass/ volume) and the point mass/volume, computing a drag coefficient would not be strictly accurate. This is because, currently, drag in Nucleus has a linear relationship with velocity, while in the real world, drag effects are relative to the square of the velocity. Part of the reason for this relationship is that the Drag currently assumes a fixed area (Or rather, the drag coefficient can be thought of as this area.). As a result, if one scales up a scene, the drag effects would seem proportionately much greater if it was non-linear because the assumed area is fixed instead of increasing as the square of the scale.

Further complicating this is a problem where the Lift attribute is using the square of the velocity, and thus becomes too high for large scenes. For example, if the scene is 100 times the scale, then the Lift value should be multiplied by 0.01 to compensate. However, one of the reasons Lift helps provide nice results is its non-linearity: when objects have more lift effect the faster they move. Unfortunately, this is also not accurate as the drag effects, being linear, do not balance the lift at high velocities, which can cause instability. Thus, Lift should be used sparingly and with care. Hopefully this situation will be remedied in a future release.

Note that the effect of Lift and Drag is directly proportional to the Mass and Air Density, with more massive objects having less drag. Thus, if one adjusts the Mass of an object, and wishes the drag to stay the same, multiply the Drag and Lift values by the same factor. For underwater effects, one can increase the Air Density. Note that this is equivalent to either lowering the Mass, or Drag and/or Lift on objects.

5.4 Tangential Drag

Tangential Drag corresponds essentially to how rough or sticky a surface is with regards to the air. When Tangential Drag is set to 0, cloth has no drag when it is slicing through the air like a knife. Most objects in the real world would have some resistance, just as the sides of a knife may stick to the material it is cutting through. When set to 0, the material is slippery, while at 1, the amount of drag is uniform in all directions in a similar fashion to a sphere. It is difficult to relate this attribute directly to a real world physical unit, but for general cloth in air, it should be a low but non-zero value. Adjusting Tangential Drag also helps provide a sense of terminal velocity for a plane falling edge on. Otherwise, the plane will continue to accelerate indefinitely as it falls.

5.5 Friction

The friction coefficient (Friction attribute) in Nucleus is designed in a 0 to 1 range, where 0 is no friction and 1 is total friction. For collisions, the friction used is the average of the Friction values for each surface. Thus, if the Friction values for the two objects were 0.0 and 1.0, the friction for the collision would be 0.5. If instead the values were 0.0 and 2.0, the resulting friction would be 1.0 (or full friction). Note that one can use coefficients greater than one as well as negative coefficients, although the resulting average will be clipped if it is greater than 1.0. However, it is not clipped (currently) if less than zero, and negative Friction values actually help add energy and increases motion, which can lead to instability, although it is useful on occasion. It is important to realize that the Friction will not always stop sliding when it is at a value of 1, although as Substeps increases, Friction is more likely to stop the sliding. In some cases, the Stickiness attribute can be used to help stop sliding without requiring higher Substeps. For strong friction, it can also help if Max Collision Iterations is set no greater than the Substeps.

5.6 Stretch Resistance and Bend Resistance

Stretch Resistance and Bend Resistance are designed in a zero to infinity range, and by default are defined relative to each link in the cloth. Also, they vary in a linear fashion unlike real objects, which have complex non-linear stretch characteristics. Thus it would be futile to try and relate real world stretch measurement values to Stretch Resistance and Bend Resistance settings. In general, for certain things like cloth, one wants little or no stretching, so it is generally a matter of finding the lowest acceptable value for Stretch Resistance and Substeps.

5.6.1 Real world simulation example

Let's say one wishes to match a real world flapping flag. First, the Space Scale attribute on the nucleus node should match the units of the flag model (0.01 for cm). Next, the Wind Speed value should match the speed of the wind in units/sec. It may help to create a simple, one polygon cloth plane with Tangential Drag set to 1 and Drag set to 1 (or a large value), then watch how fast the plane moves in the wind. Then, adjust the Wind Speed on the nucleus node to help match the motion of a bit of cotton fluff in the reference. The mesh for the cloth flag will need to be modeled with enough polygons to match the desired level of detail in the folds of the real flag, and initially be constrained in a manner similar to the real flag. At this point, the rest of the simulation is a bit less like physics and more like cooking. The required Stretch Resistance and Substeps will depend on the density of the flag mesh as well as the scene's scale and wind speed (Very large flags can actually simulate with lower Substeps because they move more slowly). Some Stretch Damp will generally be required to keep the flag from being too bouncy. The Bend Resistance would generally be low or zero, especially for a large flag made of fine silk. For smaller flags, or ones made of thicker less flexible materials, one may need more Bend Resistance. With Stretch Resistance and Bend Resistance set, one can now adjust the Drag to help match the reference. Start with Lift set to 0, and the Tangential Drag at a small but non-zero value. One can then increase the Drag (or lower the Mass) until the flag is lifted and flaps in a manner similar to the reference. To make the flapping more dynamic, one can use some Lift, but be careful not to make it too high. Lift currently is non-linear with regards to velocity, so large scale scenes need to use less. If the Space Scale is 0.01, the Lift generally should not be much more than 1/100th the value of the Drag value. If the Space Scale was 10.0 (i.e. a unit represents 10 meters), the Lift value should generally not be more than 10 times the Drag value.

6.0 Scene Scale and Units

In Maya, there are a variety of problems that can happen when one sets the units to something other than centimeters. In general, the more effects oriented features in Maya tend to fair worse in terms of using non-default units.

Maya Dynamics, such as Maya Fields do not adjust based on the unit settings. For example, the Maya Gravity field assumes that the size of a unit is a meter (default value of 9.8), regardless of the unit setting in the preferences. Nucleus was developed using the same scaling setup as Maya Dynamics to make sure both dynamics systems were similar.

Nucleus helps provide a Space Scale attribute to help allow one to compensate for different interpretation of units. If one considers a unit to be a centimeter, regardless of the actual units setting, make the Space Scale 0.01. The main effect of this is to make the nucleus gravity 100 times stronger. Note that the Space Scale on Nucleus does not adjust the strength of external fields, meaning that one needs to adjust field settings individually to compensate for scale. If one is using a Gravity Field node and one is treating units as centimeters, set the Magnitude to 980.0. With real world gravity and detailed cloth, one typically needs to increase the Substeps and Max Collision Iterations. The Stretch Resistance, Bend Resistance, and Compression Resistance values may also need to be increased.

An interesting observation is that if one has a giant character or flag, it is easier to simulate than a small one. The acceleration of gravity is proportionately slower at large scales, which is roughly equivalent to slowing down time (This is why effects artists used slow-motion for the guy in the Godzilla suit).

Note also that lowered gravity helps make air drag more noticeable. A factor that tends to even things up, however, is that fabric would be proportionately thicker at very small scales (like clothing on a mouse), and its motion is much more damped and stiff. The solver scale does not compensate for this effect and assumes that the smaller scale cloth would be made out of thinner threads. However, the effect of gravity is still very strong at the scale of a mouse. Many productions with characters at this scale slow down the overall motion and effect of gravity to make them seem more human.

7.0 Slow or Fast Motion

One can retime an nCloth cache simply by making it a Trax Editor clip and re-scaling it. This works well if one is re-timing all the related animation as well. However, one may want to speed up or slow down the simulation relative to the animation of passive objects in the scene. The attribute that controls the overall speed on the nucleus node is Time Scale. One can also adjust the frames per second (time units in preferences).

With nParticles, if one retimes a cached system, it currently does not interpolate between cached frames. Thus, it may look jerky if one slows down a particle cache too much. Note that nCloth, however, does interpolate cached frames, so one can slow down nCloth caches and still have smooth motion.

8.0 Maps and Per Vertex Cloth Properties

Cloth per vertex attributes and textures are applied as a scale factor on the relative base attribute. So, for example, values for rigidity per vertex (when Rigidity Map Type is set to Per-vertex) need to also have the base rigidity (Rigidity attribute) set to some non-zero value in order to have an effect (because zero times any value is always zero). Also, it is important to realize that per vertex values should generally be kept in the range 0.0 to 1.0. Otherwise, the simulation may become unstable since the base iterations used for an effect is determined by the base attributes. Thus, painting a Bend of zero everywhere except for a few points will not speed up the solve, because the base iteration is still defined on the Bend Resistance attribute value. Also, using values beyond 1.0 might push the solve into an unstable region (Although, one could experiment with this, and in some cases, values greater than 1.0 might still be stable yet allow for lower base attributes values and thus fewer iterations).

One exception to cloth maps being between 0 and1 is wrinkle maps, which may have numerous values, including negative ones. In fact, negative value wrinkle map values are frequently useful. For file or procedural textures, one can use a negative alpha offset to help allow for wrinkles that push in, not just out.

9.0 Presets and Scaling Relation

If one used an nCloth preset (e.g. burlap), the Scaling Relation attribute is set to Local Space. (nCloth presets currently use this setting.) Local Space essentially helps normalize attributes, such as Stretch Resistance and Bend Resistance, so that the material behaves the same for different resolution meshes. The default setting of Scaling Relation is Link, which instead helps define these attributes relative to each link so that a high resolution mesh appears to have more stretch than a low resolution mesh. Note that with Scaling Relation set to Local Space, the performance difference between low and high resolution meshes will also be greater, because there are more iterations required to preserve the behavior as the resolution increases. One thing to be aware of is that when Scaling Relation is set to Link, the values for Stretch Resistance and Bend Resistance indicate the number of internal iterations per step.

The different elements of the cloth solve iterate at different rates and the solver interleaves these iterations. For example, if the Max Collision Iterations is set to 4 and the Stretch Resistance is set to 20 (with Scaling Relation set to Link), then there will be one collision computed every 5 stretch iterations. Collide iterations are generally more expensive than stretch iterations, but one generally needs one for at least every 8 stretch iterations. The substeps help provide a minimum level of iteration as well as help allow the solver to take smaller steps. In general, the quality of the simulation will improve with increasing Substeps, and usually will take longer to compute. Each substep Nucleus recomputes bounds, and iteration below the substeps will now be done at the substep level. For example, if the Bend Resistance is 1 and the Substeps is 10, Nucleus computes 10 iterations of bend per step (The intensity of bend resistance per iteration is proportionately less, so the overall effect is roughly the same.).

When the Substeps are low, the collisions are more likely to fail and low Max Collision Iterations can also result in stretching where the mesh is in collision, even if the Stretch Resistance is set to a very high value. As a general rule of thumb, set the Substeps to values that are at least 1/10th of the Stretch Resistance value, and set the Max Collide Iterations to a value around 1.5 times the Substep value (note that internally the collide iterations are at least the Substep value even if the Max Collide Iteration attribute value is set lower than Substeps).

The current presets are generally on the low quality side to keep performance optimal. In many cases, one may want to increase Stretch Resistance. The true stretch resistance of materials like silk is so great that it would take a very long time to simulate. It is generally sufficient to simply not see noticeable stretching for computer graphics. Also, one can use Stretch Damp, which is preferable to the global Damp in many cases (The nCloth Presets were made before Stretch Damp was added to Maya.). Also the presets do not set the Substeps attribute, which often needs to be higher than the default settings.

10.0 General nCloth character setup

Setup one's character so the cloth will respond correctly to poses and translation of the character: first do a bind skin (Skin > Bind Skin) on one's cloth to one's skeleton before making it nCloth. The nCloth will help deform and translate with one's character when one sets its start position. Note that, by default, the rest shape of the nCloth is set to the start shape of the nCloth. If one has posed the character, then the start nCloth shape may be stretched heavily in places, and thus would not be a good rest shape. One can set the rest shape to be the non-deformed cloth as follows:

1. In the Hypergraph, list Input and Output connections and find the mesh node for the cloth that is just upstream of the bind skin deformation. This will have Intermediate Object set to on.

2. Turn Intermediate Object off, and one can see the rest cloth shape in addition to the posed cloth.

3. Select the nCloth shape node in addition to this mesh, and then select nMesh > Rest Shape > Connect Selected Mesh to Rest Shape. If one sets the start pose to something really stretched, the cloth will pull back when one simulates. One can then do relax initial state (nSolver > Initial State > Relax Initial State) to get a natural looking start position. Note that the location of the rest shape is not important, only the shape. Also, one can turn Intermediate Object back on when one is done to hide the rest shape.

If one doesn't need to pose the character, but simply translate the position, then one could simply parent the cloth input mesh to the base character position. Note that one needs to parent the mesh, not the nCloth node. Also, if one used World Space Output when one created the nCloth, one needs to take care to parent the input and not the output mesh. At any rate, if one used the bind skin approach above, there should be no parenting required.

Also see the tutorial at the following location: **http://area.autodesk.com/index.php/blogs_duncan/blog_detail/basic_cloth_on_character_setup.**

11.0 Effects

11.1 Shattering Glass

To help create shattering glass, one can simply make the object nCloth, and then create a Tearable Surface constraint, selecting either the full object or just the edges one wishes to be breakable. (If one has a polygon cube, increase its subdivisions to have lots of breakable bits, as the Tearable Surface constraint breaks along the polygon edges.) If desired, one can select edges before creating the Tearable Surface constraint, and the object will only be breakable along those edges.

However, the one of the best ways to do this is to first split up one's geometry the way one wishes it to break. This is a matter of detaching edges, but also keeping everything part of the same mesh (the breakable parts should show up as boundary edges). Then, make the object nCloth. At first, the object falls apart, but one can create a constraint and adjust Glue Strength to hold the pieces together. A very fast way to do this is to select the mesh, create a Component to Component constraint, and then set the following:

- Constraint Method to Weld
- Connection Method to Within Max Distance
- Max Distance to 0.

This will help constrain between the overlapping boundary points. However, there will not be bend resistance across the joint. For resistance, one needs to create a Weld Adjacent Borders constraint. One can then use the Bend Strength setting on the constraint (Be aware that bend constraints are currently VERY slow to compute for a large number of edges.). To help make the surface look continuous until it breaks, do Edit Mesh > Merge and Normals > Soften Edge. (This is the same setup that is created by the Tearable Surface constraint menu.)

For glass, one will need very high Stretch Resistance, Compression Resistance, and Bend Resistance values. Also, the Substeps should be increased. Be mindful of the self collision method and self collision thickness, as thick self collisions can slow down the simulation.

To get the thickness of the glass simply extrude after either creating the Tearable Surface constraint or manually making the tearable setup with Edit Mesh > Merge. With the extrude downstream of the polyMergeVert node when the mesh breaks up, one gets cracks in the glass.

For very stiff objects like glass, an alternative constraint might be a Transform constraint with the Glue Strength set such that each piece would stay in place until knocked by an object. However, if one does this, one can have chunks of the window break off yet hold together, which could look unnatural.

11.2 Tight clothing and shrink wrap

One can shrink nCloth by lowering the Rest Length Scale attribute. For a lot of shrink, one might want to animate the attribute setting, gradually reducing its value from 1.0 to avoid a violent snap. If the garment is stretchy and tight fitting like nylon stockings, one can also lower the Stretch Resistance while adjusting Rest Length Scale. Otherwise, the stretch may fight too hard against collisions.

For shrink wrap, one can lower the Rest Length Scale, and at the same time, make the nCloth pressure negative, simulating vacuum packing.

11.3 Attaching buttons to shirt

To make the task of adding buttons to an nCloth shirt easier, one can use a parent-to-surface script. One can access the parentToSurface.mel script file from Duncan's Corner at the AREA at the following location:

http://area.autodesk.com/index.php/blogs_duncan/blog_detail/parent_to_surface_script.

Look under Related Materials for the parentToSurface.mel script file.

This script sets up hair follicle nodes on the mesh to attach objects. Hair follicles are good for deriving a full transform with good rotation from a point on a mesh.

For buttons that actually collide with the shirt and not just move with it, one can create a single quadrangle plane for each button (each should have different UVs). Make all the buttons a single nCloth object with high Stretch Resistance, Compression Resistance, and Bend Resistance values, but no Rigidity. The cloth thickness can be set to help match the buttons. Use a Component to Component constraint to help simulate the button thread connections, and then use the parent to surface script to parent the detailed buttons to the simple planes.

11.4 Underwater Shirt

When a shirt is underwater, the drag acting on the shirt is much higher. To produce this effect, one can simply increase the Air Density on the nucleus node, or increase the Drag and Lift on the nClothShape node. The effect of passive objects on the water flow is important: On the passive (nRigid) objects, make the Air Push Distance about one or two times the size of the objects. The Air Push Vorticity (added in Maya 2008) can be used to simulate water flowing around the passive objects without needing a fluid simulation. Also, if the cloth has neutral buoyancy, set the Gravity to 0.

One could also apply a fluid effect as a field on the cloth, although the fluid effect will tend to keep adding velocity like a force, rather than behaving like an air or drag field. However, as long as the nCloth Drag is relatively high, this approach should work.

11.5 Simulating only pant leg bottoms

In some cases, one may want to use a Bind Skin animation for the upper part of pants while dynamically simulating just the bottom of each pant leg. This setup can run much faster than simulating the entire pant.

One can also try painting Input Attract, making it 1.0 everywhere except the pant bottoms. Unfortunately, this does not speed up the solve.

An alternative workflow that may simulate faster is the following:

1. Before creating nCloth do a bind skin (Skin > Bind Skin) on the pants (if one hasn't already), and then do a Mesh Separate at the knees (with history) so there are now two meshes.

2. Make the bottom part of the pants nCloth, and on the nCloth set Input Mesh Attract to 1.3 or so. On playback it should now simply follow the skeleton deformation.

3. Paint the Input Mesh Attract (nMesh > Paint Vertex Properties > Input Attract) on the pant legs such that the very top is white, with black everywhere else. The pants now playback with nCloth dynamics.

4. Rejoin the top and bottom pant legs by doing Mesh > Combine, Edit Mesh > Merge, and Normal > Soften Edge (all with history).

5. Turn off Self Collisions on the nCloth if they are not needed.

Another basic technique to speed the simulation up is to do a mesh smooth (Mesh > Smooth) downstream of the cloth, rather than simulating the detailed mesh. Or, one can simulate on low resolution cages that are then used as wrap deformers on one's final mesh.

11.6 Rebuilding constraints for mesh topology changes

If one has explicitly defined a constraint to connect particular vertices, edges, or faces, the constraint may no longer work properly if one changes the topology of the nObject in some fashion, such as substituting a higher resolution mesh for a cloth object. This happens because the indices of the components may now be different due to the topology change. To fix this, one can reselect the components on the object that was modified as well as the constraint, then select nConstraint> Replace Members. (If there is only one constraint for the object, one need not select it.)
It is often possible, however, to define the nComponents in an implicit fashion so that it will still work, even though the topology of the constrained object has changed. If, when creating a constraint, the selection consists of either objects or components (vertices, edges or faces), then the resulting constraint will be topology independent. This translates into the Elements attribute on the nComponent node being set to either All or Borders. However, one may still not wish to connect to all elements of the nComponent. There are currently three primary methods one can use to implicitly define subsets of components to constrain:

Method 1. The elements attributes may be set to Borders if only border components are desired.

Method 2. Set the constraint Connection Method to Within Max Distance to filter out connections based on distance. One can then use Within Max Distance and set a Connection Density value to define the connections formed.

Method 3. Either use nConstraint > Paint Properties by Texture Map> Strength or assign a 2D texture to Strength Map on the nComponent. This will not remove constraint links, but it will help remove their effect wherever the texture is 0. Note that one needs to use a texture map, not paint per vertex values because per vertex values would be affected by topology changes to one's mesh.

Note that the Weld Adjacent Borders constraint uses both methods 1 and 2 above. It helps provide a resolution independent method of stitching seams between cloth objects.

In rare cases, one may be able to get some topology independence of constraints by using sets (Use Sets), although this will only work with very specific construction history. Most changes to construction history will not be properly compensated for by sets. Also, sets add a great deal of complexity to the constraint setup so the Use Sets option for constraint creation should be left off unless one is very advanced. Also note that for nComponents, the sets used must be all of a particular component type. For example, one should not mix edges and vertices in the same set.

12.0 Maya Caching of Nucleus Objects

Information about Maya nCaching workflows are already covered in the Maya documentation—one can create them, delete them, blend them, modify them in the Trax Editor. Basic cache file format is covered too—Cache files have an xml header, and one of more .iff files that contain the actual cached data. For the details of the actual cache formats, refer to the Python™ scripting examples that demonstrate reading a geometry cache—see the cacheFileExample.py and cacheFileConverter.py scripts in the devkit/pythonScripts area.

In this section of the document, the following nCaching topics are discussed:

- Similarities and differences between the different nCache files—when one can and can't use them together
- Cache node connections, channel names, interpretations and troubleshooting a messed up cache
- Creating caches in batch
- Optimizing one's playback from the cache (or not)
- Some known bugs in Maya 2009, and how to work around them.

nCloth caching, nParticle caching, geometry caching (and fluids caching) share the same cache format and cache nodes, and the data can be shared to a certain extent. However, there are some fundamental differences between the different caches that one needs to be aware of when using different caches in the same scene.

12.1 Similarities and differences between file data

12.1.1 So what data streams get cached for which objects?

Geometry cache: positions, local space only

nCloth cache: cached from the solver internal state

- Positions – local or world space
- Positions and Velocity – local or world space

nParticle cache: cached from the particle attribute state

nParticles caches have: particle Id and AgePosition – local space

- Position and Velocity – local space
- Dynamics and Rendering – the functionality above as well as mass, lifespanPP, radiusPP, opacityPP, rgbPP, and the sprite placement attributes if they exist.
- All – the functionality above as well as other dynamic PP attributes that have been created

Note that in future additional attributes may be added to the cache as needed.

12.1.2 Cache layout

File per frame or single cache? File per-frame helps allow one create larger caches and avoid the 2G iff file limit, and it also helps allow one to replace cache frames. Single file is tidier, and makes it easier to package up a cache.

nParticle caches require file-per-geometry, and the UI will turn it on internally, even if one didn't specify it. So, if one caches nCloth and nParticle objects at the same time, one will still end up with file-per-geometry. If one tries to force nCloth and nParticle objects into one file by using the cache command directly, one will be sorry. Geometry caches and nCloth caches can be saved either way, but there is currently a bug with position and velocity caching that may force one to fix some of the connections manually afterward. If one plans to create caches and then assign them to different objects later on, having multiple object data in a file will lead to confusion. If one plans to experiment with blending between caches, one should consider ahead of time whether or not one wishes to do this per-object before deciding on how to cache. (As for fluids, the UI doesn't let one do geometry per cache, but the cache file command will allow one to do it.)

Float or Double? Geometry caches help allow one to store points as either float or double. nCloth data does not give one the option and always stores as float, while nParticles don't give one the option and always store as double. Why does it work this way? If the geometry cache positions are doubles, one is free to choose to save them as floats to optimize cache file size, if float precision is good enough for their purposes. For the nCloth cache, float was chosen because the internal solver state is float, and this avoids introducing errors by needless conversion back and forth between floats and doubles. For nParticles, total state is defined by the particle attributes since expression can be applied post-solve to get the final state, and these are applied to the attributes, which are doubles. But they are saved as float anyway by default, to keep the file size down.

Fortunately, the cache command and cache node will swap back and forth between float and double, and the nCloth node will accept float or double data.

12.1.3 Assigning caches between types

nCloth cache as geometry cache – works as long as the data is in the same space

Geometry cache as nCloth cache – works as long as the data is in the same space

nParticle cache to nCloth or geometry – works but one may have to remap the channel name from position to positions - see section on connections below

One can't assign an nCloth or geometry cache to nParticles, because particles need Age and ID. One can't exchange a fluids cache for any of the other caches in a meaningful way.

12.2 Connections and Channel names

12.2.1 Channel name conventions

For a single channel cache, whether there is one object or there are multiple objects, the channel is the object, while the interpretation is the attribute name. An example of the cache description file is as follows:

```
<channel0 ChannelName="nClothShape1"
ChannelType="FloatVectorArray" ChannelInte
rpretation="positions" SamplingType="Regular"
SamplingRate="250" StartTime="250"
EndTime="6000"/>

<channel1 ChannelName="nClothShape2"
ChannelType="FloatVectorArray" ChannelInte
rpretation="positions" SamplingType="Regular"
SamplingRate="250" StartTime="250"
EndTime="6000"/>
```

For a multiple channel cache, the channel name is nodeName_attribute and interpretation is the attribute name. An example of the cache description is as follows:

<channel1 ChannelName="nClothShape2_positions" ChannelType="FloatVectorArray" ChannelInterpretation="positions" SamplingType="Regular" SamplingRate="250" StartTime="250" EndTime="6000"/>

<channel2 ChannelName="nClothShape2_velocities" ChannelType="FloatVectorArray" ChannelInterpretation="velocities" SamplingType="Regular" SamplingRate="250" StartTime="250" EndTime="6000"/>

If one attempts to cache a multiple channel and single channel object together in the same cache, one may end up with both, and then have to fix the connections on the cache node manually.

12.2.2 Data connections – per channel or single connection?

The geometry, nCloth, and fluids caches have per-channel connections, while the nParticle cache has one large connection which has all the data. Both have advantages. With a per-channel cache, one can select which piece of data one wants to read from the cache and connect to each attribute. This means one doesn't need to read the data one doesn't want. One cache node can be connected to multiple objects. Data can be blended per channel, as long as the caches contain the same data; it's very convenient. However, one cannot attach a new file that has different data. If one wants to add a new type of data, one needs to add new attributes.

Since nParticles could have anywhere between three and many cached attributes, it was preferable to be able to connect them with one connection. So, if one browses for a different particle cache that has a different set of cached attributes, one can leave the connection the same. Actually, one cheats and leaves the positions connection there as well, since much of the nCaching UI counted on the presence of that connection.

On the cache node there are four attributes of interest for how data gets passed between the cache node and cached node:

inRange –Is set if there is data on some channel in this cache at this time. Normally, if there is data for one channel at this time, then there should be data for all of the channels. However, if data was appended with different cache settings than the cache was originally created with, it may be possible for some channels to have a different range. If one suspects that something like this is happening, look at the XML header for the ranges of each of the channels. Of course, if the cache was created with an application other than Maya, it's the one's responsibility to make sure that the headers reflect the actual content of the files.

channel – If there is more than one channel in the cache, and the data is individually connected per channel, this attribute is used to select which data is going out on which connection. For example, if one has an nCloth cache with velocity:

channel[0] is set to "nClothShape1_positions"

channel[1] is set to "nClothShape1_velocities"

outCacheData – Individual data channels are passed through elements of this attribute. – in the above example:

nClothShape1Cache1.outCacheData[0] is connected to nClothShape1.positions

nClothShape1Cache1.outCacheData[1] is connected to nClothShape1.velocities
So, if one initially had a cache with only positions and one browsed to a cache that contained velocities too, one would need to manually set up the velocity channel. For example:

setAttr -typ "string" nClothShape1Cache1.channel[1] "velocities";

connectAttr nClothShape1Cache1.outCacheData[1] nClothShape1.velocities;

For nParticle cache connections, a single large connection is used instead:

outCacheArrayData - this attribute connects to nParticleShape.cacheArrayData

At the moment, there's only support for one nParticle system's data per cache. outCacheArrayData is a single attribute and not a multi, and the cache channels are collected in that attribute, using the interpretation as the key. If the interpretation does not match the name of a per-particle attribute, the data will not be read back into the nParticle node.

The positions connection has been retained because much of the nCloth UI checks that connection. If there are problems when attaching an existing cache, one may need to check that the right channel (i.e. the position channel name from the xml header) is selected.

12.3 Creating caches in batch

Command documents for the doCreateNClothCache command have not been created. Use the doCreateNclothCache command for both nCloth and nParticles. For example:

doCreateNclothCache 4 { "2", "1", "12", "OneFilePerFrame", "1", "","0","","0", "add", "0", "1", "1","0","1" } ;

One needs to select the objects to cache first, and one should ALWAYS do cache per geometry for nParticles.

global proc string[] doCreateNclothCache(int $version, string $args[])

Description:
Create cache files on disk for the select ncloth object(s) according to the specified flags described below.

current version = 4

$args[0] = time range mode:
    time range mode = 0 : use $args[1] and $args[2] as start-end
    time range mode = 1 : use render globals
    time range mode = 2 : use timeline
$args[1] = start frame (if time range mode == 0)
$args[2] = end frame (if time range mode == 0)

$args[3] = cache file distribution, either "OneFile" or "OneFilePerFrame"

$args[4] = 0/1, whether to refresh during caching (not so interesting in batch)

$args[5] = directory for cache files, if "", then use project data dir

$args[6] = 0/1, whether to create a cache per geometry

$args[7] = name of cache file. An empty string can be used to specify that an auto-generated name is acceptable.

$args[8] = 0/1, whether the specified cache name is to be used as a prefix

$args[9] = action to perform: "add", "replace", "merge" or "mergeDelete"

$args[10] = force save even if it overwrites existing files

$args[11] = simulation rate, the rate at which the cloth simulation is forced to run

$args[12] = sample multiplier, the rate at which samples are written, as a multiple of simulation rate.

$args[13] = 0/1, whether modifications should be inherited from the cache about to be replaced.

$args[14] = 0/1, whether to store doubles as floats (cloth is all floats anyway, if one does this for particles, one might lose precision)

12.4 Optimizing playback from the cache (or not)
Before one creates one's cache, one should think about how one will be playing it back. The big question is: Does one plan to append or resume simulation? Or, does one plan to just play back off the cache, or perhaps blend it with other caches?

When playing back off the cache, the solver state is updated at every frame, as it's not actually known when the end of the cache will be reached. And, if only some of the objects attached to one's solver are cached, their state has to be updated in order to be able to solve the rest correctly. If one knows one isn't going to do either of the above, one can disable the active cached nObjects and the nucleus node in order to help get the best performance from the cache playback.

On the other hand, if one wants simulation to resume off the end of the cache, there are a few other things to consider. One will need to make sure that one's saving out enough data to the cache to resume simulation, and one will need to be aware of the problem areas for getting an exact match when playing back. Consider the following:

nCloth - Make sure one caches position and velocity (Select Position and Velocity for the Cacheable Attributes attribute option.). If one only caches position, then the velocity will be the average velocity across the frame, which will not be correct for vertices which collided during the last cached frame. Note that there can be precision errors with a local space Nucleus object cache: use a world space nCloth if exact precision is needed. Note that if there is a self-collision failure during the last cached frame there may be visible differences when resuming simulation. So, make sure that the quality settings are high enough to prevent self collision failures if exact precision is needed. Several other factors may also cause differences when resuming simulation using: Rigidity, Wind Noise, Wind Self Shadow, and the constraint Connection Density Range. In these cases, the simulation quality may not be any worse but the results may be different.

nParticles - Depending on one's expressions and events, one may need to cache either Dynamics and Rendering or All. Select either Dynamics and Rendering, or All for the Cacheable Attributes attribute option to get the simulation to resume correctly. The cache is currently created with "save double as float" on by default. This should not cause significant differences for the dynamic state, since this is computed as float. However, expressions computed on double attributes may produce different results. If this is causing visible errors, one may also need to invoke the doCreateNClothCache command manually (see above) to force the attributes to be saved as double. This means that one's cache will be twice as big – it's one's own decision. Note that the random emission seeds are not currently cached, so while the simulation will help match for particles in existence in the last frame of the cache, subsequent emission may be different.