Masters Thesis

PHYSICALLY BASED ELASTIC BEHAVIOUR

Implementation of a Partial Differential Equation solver in Maya using the API.

Technical report.

Javier Romero
MSc Computer Animation
d1141914
September 2005

INDEX:

1.- Abstract.

2.- Introduction.

2.1 .- Outline.

3.- Previous work.

4.- Physics of elastic behaviour.

4.1.- Dynamics of deformable models.
4.2.- Dynamic PDE equation.
4.3.- Solving the equation.

5.- Tool implementation.

5.1.- Framework and data flow model.
5.2.- Object orientation.
5.2.1.- Elastic node.
5.2.1.1 Functions.
5.2.1.2 Attributes.
5.2.2.- Elastic Force Node.
5.2.2.- Initializing the plugin.
5.3.- Solving the mesh.

6.- Simulation examples.

7.- Conclusions and future work.

8.- References.

1.- Abstract.

Modeling and animation of certain natural behaviours, like those found in elastic or inelastic bodies, can be a very tedious task if done by keyframed methods. Using the physical laws that define these behaviours, can lead to more realistic results by controlling the parameters of real life objects. In the past some implementations have been made using constraint methods or energy minimization algorithms. In this thesis, a new animation technique based on a existing fourth order dynamic partial differential equation (PDE) is described and implemented as a usable Maya tool, in order to depict elastic and plastic behaviour in objects.

2.- Introduction.

Many times, research groups at universities ara far away from production pipelines. Several methods developed on this highly academic teams could be used on productions that would benefit greatly of their studies instead of starting from scratch.

With this project a production ready tool is presented based on an existing research on physics of deformable models, in form of a Maya plugin. From the experience developing the tool, conclussions about future methods of solving partial differential equations, data interface between the user and the software and possible bottlenecks to avoid, are presented.

2.1 .- Outline.

First , an overview about previous work and key topics on deformable models for computer graphics and animation is discussed. Secondly, an introduction to the theoretical and technical aspects that drive elastic beaviours in nature and in previous approaches to the field is described and compared with the method here implemented. Thirdly, a description about the tool and the general framework it follows along with the different parts that it is comprised of, finishing with some examples of deformations generated with the tool. At last, several conclusions are presented and discussed as future work.

3.- Previous work.

Deforming objects to create natural and realistic shapes and movements is a topic that has been researched widely since the beginning on the computer animation field. It can be achieved by direct keyframing of the vertices involved in the deformation, but this is quite consuming and the effects are hard to be made convincing.

Traditional methods to animate deformed objects include the so called bijective space mapping, where a set of points are transformed into a function deformed space. As presented by Barr[2], once the involved points are projected into the deformed space, they are returned to their original space but keeping the deformed positions. This method works very well for global and local deformations. A deformation is said to be global when it affects all the vertices ot control points of a shape, or local when only a number of them are deformed. The visual complexity and realism resides purely on the skills of the animator.

In [3], Sederbergh and Barry presented a technique to defrorm free form surfaces. The method is more powerful than the one presented in Barr, because the animator can achieve more complex forms and it is more intuitive.

Other approaches use real world physics in the generation of shapes and dynamic behaviour. This is the case of physically based animation. The simplest of them is the spring-mass model. In it each vertex of a shape is modelled as a point mass and each edge as a spring, with a resting length equal to its original length. As external forces are applied to specific vertices, they will be displaced inducing spring forces which will produce a propagation of the force resulting in more displacements. The final result will be a deformed shape that, depending on the mathematical model or the points of application of those forces, can describe cloth, flexible objects, soft bodies, etc.

In [7] the concept of a system´s energy is introduced. In this context, energy is free to be defined in any sense it serves the task. A set of functions are defined in order to pin objects together, restore the original shape, etc. By minimizing this functions, a resting shape can be found and achieved then. The drawback on this method, like on the spring-mass model, is that more than one resting shape can be achived and it may not be the desired.

Other authors use mathematical models based on simplifications of the elasticity theory. That is the case of Terzopoulos [6] , who describes elastic behaviour in static shapes by simulating physical properties such as tension and rigidity of a body. Also dynamics of those objects are introduced, compatible with rigid body motion, such as mass and damping.

In You et al. [19], a simplification of the force produced by the elastic properties of matter is employed, by proposing a dynamic 4^{th} order partial differential equation. The motivation behind it is that a 3 dimensional shape can be described in form of a PDE subject to boundary conditions, and it is extended for time dependant 3D deformable surfaces by introducing a new variable t.

In this thesis and implementation in Maya of this procedure is presented as a production ready tool used to describe dynamic elastic behavior on polygonal shapes.

4.- Physics of elastic behaviour.

This section deals with and explains the required technical background associated with the elastic deformable models and the methods implemented in this tool from [6] and [19]

A deformation is named elastic when the deformed or reference shape restores itself completely upon removal of all the external forces. It is possible to quantify elastic restoring forces in terms of potential energies of deformation: an elastic model stores potential energy during the deformation process and releases it entirely as it recovers the reference shape.

On the other hand, plastic behaviour is the one shown in obects that, upon removal of those external forces, do not return to the starting shape, it remains deformed. The potential energy of deformation is dissipated in form of heat, etc.

4.1.- Dynamics of deformable models.

In [6], a mathematical development to obtain the equations of motion governing the dynamics of deformabele models under the influence of external forces is presented.

Following it, for any given 3D solid, a point from it can be represented using its local coordinates. This positions are given by a time varying vector valued function of the material coordinates.

$$r(a,t) = [r_1(a,t), r_2(a,t), r_3(a,t)] \ ; \qquad eq.1$$

So, for an initial resting shape at time $t = 0$;

$$r(a,0) = [r_1(a,0),\ r_2(a,0),\ r_3(a,0)] \ ; \qquad eq.2$$

Given that, the equations governing the motion of a deformable body can be put in Lagrangean [6] form:

$$\rho \frac{\partial^2}{\partial t^2} + \eta \frac{\partial}{\partial t} + \frac{\partial \xi(r)}{\partial t} = f(r,t) \ ; \qquad eq.3[6]$$

Where $r = (a, t)$ and represents the position of the point a at time t. (a) is the density of the solid at point a, (a) is the damping density and $f(r, t)$ represents the sum of all external forces applied at a at time t. (r) is a function that measures the net instantaneous potential energy of the elastic deformation of the body.

The three terms represent all the internal forces generated by those physical properties:

$$\rho \frac{\partial^2}{\partial t^2} \ ; \quad Inertial\ force\ \ due\ distributed\ mass\ of\ the\ body \qquad eq.4$$

$$\eta \frac{\partial}{\partial t} \ ; \quad Damping\ force\ due\ dissipation\ of\ the\ force. \qquad eq.5$$

$$\frac{\delta \xi(r)}{\delta t} \ ; \quad Instantaneous\ potential\ energy\ of\ the\ elastic\ deformation\ of\ the\ body. \qquad eq.6$$

$$f(r,t) \ ; \quad Net\ external\ forces\ at\ r\ on\ time\ t \qquad eq.7$$

4.2.- Dynamic PDE equation.

Eq.4 and Eq.5 are simple operators to deal with using standard numerical integration techniques, but Eq.6 is a complex strain energy function that is approximated in [6] by a vector expression that describes the elastic properties of solids.

In You, this operator is substituted with a set of shape functions of parametric variables u and v on a given point r at time t.

The original motivation for this equation comes from the fact that the representation of a 3D static surface can be regarded as the solution of a PDE subject to boundary conditions. So, for dynamic geometric modeling, where the surfaces are time dependent, the equation presented by You is:

$$\left\{ b_i(u,v,t)\frac{\partial^4}{\partial u^4} + c_i(u,v,t)\frac{\partial^4}{\partial u^2 \partial v^2} + d_i(u,v,t)\frac{\delta^4}{\partial v^4} + \rho\frac{\partial^2}{\partial t^2} + \eta\frac{\partial}{\partial t} \right\} r_i(u,v,t) = 0 \quad ; \qquad eq.8[19]$$

With $i = (1, 2, 3)$, being the three positional components of the particular point in the solid to deform.

The parameters $bi(u, v, t)$, $ci(u, v, t)$, $di(u, v, t)$ are shape functions of u , (u,v) and v directions respectively, and are related to the deformation coefficients inherent to all matter: Young´s module and Poisson´s ratio.

As in eq.3 , density and damping are kept in order to produce forces along time in this formulation. For this scope, the assumption of constant values on all the parameters on all directions is made: the tool deals with isotropic materials, despite the fact that they can be animated, as it will be described later on.

When the shape is in resting state, it equals zero, which is an equilibrium state. But when a force is introduced on the system, the pertinent component of it must be used to solve the equation for that particular state, resulting in a new resting , but deformed, state. Until the removal of that force, which provokes the shape to come to rest again.

4.3.- Solving the equation.

A differential equation does not have a solution, only an approximation that depends on many factors: number of iterations on a given interval, method used to solve the system of equations, etc. The way to solve a PDE is using a solver or ODE solver, which is a program that generates that approximation to the suggested equations.

The resulting solutions can be very different based on the number of iterations made on the unit time, and this is directly related with the speed. More iterations mean more time to solve the equation on the time unit, but more accuracy. If the approximation is made to a higher degree derivative, the time step can be bigger, resulting in a more optimized method: faster and more accurate.

In this case, eq.8 is discretized by applying a finite difference approximation (FDM), using central operator, transforming the PDE into a system of ordinary linear equations, that will be later on solved using standard techniques.

Due to the nature of the FDM , the systems must define a finite difference mesh. Its values are defined in a regular M x N grid on the (u, v) parametric space on the which the surface is defined, with horizontal and vertical internode spacings h equal in both directions. Each node or vertex has its own (Px, Py, PZ) values in space.

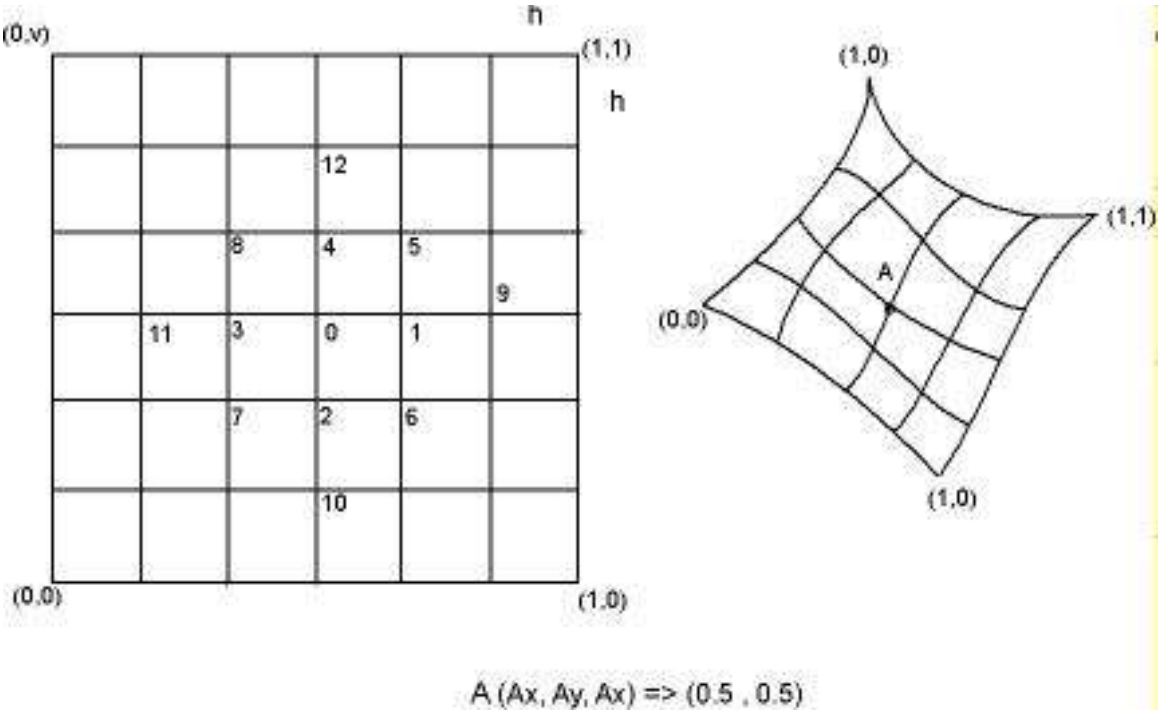


Fig 1. Relation between the position of vertex a and its (u,v) coordinates on the MxN grid

Also the nodes must be indexed using some rule in order to the solver be able to specify what the neighbour nodes are. Using this method sets restrictions on the mesh type and in the indexing of its vertices in Maya. They must grow in number as the u and v values increase. Havig stated those conditions, a typical finite difference mesh is defined.

Once discretized, a system of linear matrix equations is generated for the (x, y, z) components and solved along time. Considering that one second has 24 frames for film resolution, which is the number of times the PDE Solver has to output its results, the system is solved dt times on the unit time. When total time of iterations equals 1/24, the solution is output.

5.- Tool implementation.

The implementation of the existing mathematical model from REF into Maya meant to adapt both and meet in the middle. The development of the interface between the software and the solver was not a easy task. The package has its own methods, functions and workflows, which were adapted to fit the simulation scheme. And vice versa.

5.1.- Framework and data flow model.

Maya was used as the central data gathering and user friendly human interface. The correct definition of the data flow system between the software, the mesh, the user and the solver engine means the success of the tool. All the information needed for the FDM is compiled from different sources from the package and sent to the PDE. Once it is updated, it must be maintained in the proper place: the result is output to the deformed mesh, back to the solver for a new iteration, or even destroyed.

Given this , the general framework looks as follows:

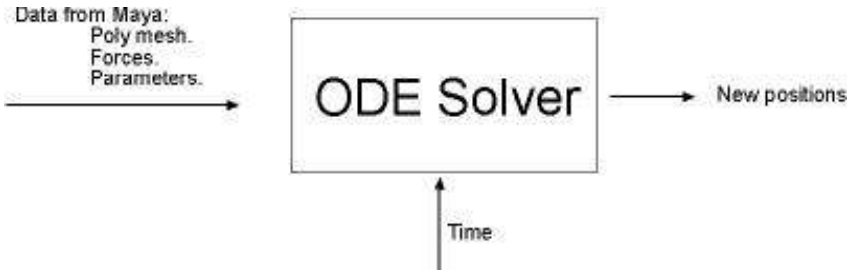


Fig 2. Function of a ODE solver in a computer animation program.

It is key to understand how Maya treats the information. Data is transferred between a network of nodes in the Dependency graph. A node holds any number of slots that contain the data used in the computations. When interconnecting several nodes, complex results can be achieved, so designing the proper data flow model between the nodes and what is the state of this data is a key point on the development of any plugin.

In this case, the structure generated is built around a supernode called elasticNode. When any node is being asked to perform its computation, by either a call from a change on the time slider, or because another node requires its output, it checks all its inputs and, if they need to be updated, it does so. This way only the required data is computed on the required time, resulting in faster evaluations.

In the case of the elastic node, it will be only updated when time changes. When a new frame is computed, the node queries to all the nodes connected to it as inputs to send it the data. Once it is gathered, it is sent to the PDE Solver, which updates the surface back into the node, and sends it to the output plug.

If the data is not updated, or if a plug sends the information when it is not required or even if the operations are performed over the wrong block of data, the system would produce unexpected results, or even become unstable. Special care is taken with the mesh data:

For frames previous to the one chosen as start of the simulation, the ouput mesh data (block containing all the mesh related information, vertices, positions, uv coordinates, etc.) is the same of the input. But when the mesh has been updated, because the simulation has started, the input mesh is not longer valid: it would lead once and again to the same result. In order to make the mesh updating work, the solved data is copied inside the elastic node, and sent back to the solver, resulting in animation of the mesh. Of course, this is the same data block sent to the input of the resulting mesh.

The force node is responsible of defining the deformation and force regions and the vertices it affects. Also, the direction and magnitude are specified here by the animator. Time is an existing node in Maya, that outputs the value of the current frame, and, by doing so, is responsible of the updating of all the system.

5.2.- Object orientation:

Next , a brief overview on the functions and attributes the nodes have is presented. A more in depth discussion can be found in 5.3 and 5.4 .

5.2.1.- Elastic node:

As mentioned in 5.1, the elastic node is the central node of the tool. It is responsible of gathering, filtering and parsing the appropriate parameters to the PDE Solver in order to update the resulting deformed mesh. It is created derived from the MpxNode Maya proxy class, which contains the compute() function that updates and queries the required data. It also contains the PDE Solver and the set of functions related to it.

5.2.1.1 Functions.

`static MStatus initialize () ;`
This function is responsible of initializing and defining the attributes and when they must make the compute function update the data.

`static MStatus *creator () ;`
Calls the constructor of the class.

`virtual MStatus compute() ;`
The main function of the node, it is responsible of gathering, filtering and maintaining the information up to date between the user, the scene and the solver.

`void PDESolver() ;`
The solver uses standard C++ data types that must be generated from Maya objects. Once the data is ready for every component (x, y, z), it calls the EqSolver. It returns the updated (x` , y` , z`) that are sent back into the solved mesh data block. It also updates the velocities of the vertices affected on that particular frame.

`void EqSolver() ;`
Generates the linear matrix system of equations for every component, and updates the data, including boundary nodes.

`int  ciggj () ;`
Solves the sparse linear equation.

This functions are discussed more in detail on section 5.3 .

5.2.1.2 Attributes:

Depending on the origin of the data the node stores, a distinction in the attributes can be made.

a) User input / defined attributes; related to the elastic and dynamic parameters from eq.8. These are the animatable parameters.

b) Mesh storage attributes; responsible of the storing and handling of the mesh data block. Not visibles by the user.

c) Internal attributes; to be used between the solver and compute functions and connections from and to other nodes. Not visibles by the user.

a) User defined attributes:

In order to create a powerful tool it has to be as much parameterized as possible, without makig the user feel lost. This gives more possibilities when the animation process comes. As shown in eq.8, the attributes to animate are easy to define: having a direct control over the equation gives more control over the final result.

These manipulable attributes are:

- Dynamic properties: Define the movement but not the shape of the body during simulation:

`static MObject density ;`
Defines the force related to the inertia of the body. With bigger inertia, more force is needed to move the body, but also harder to stop. Direct effect on the acceleration.

`static MObject damping ;`
Determines the attenuation of the movement along the body. Related with the velocity.

- Elastic properties: Define the shape of the deformation, but not the dynamic behaviour.

`static MObject parameterB ;`
The shape of the deformation along the U direction is controled by this attribute.

`static MObject parameterC ;`
Controls the curvature of the deformation along U and V. Also the resistance.

`static MObject parameterD ;`
The shape of the deformation along the V direction is controled by this attribute.

- Time related attributes:

`static MObject deltaTime ;`
The user can employ this attribute to determine the time step for the PDE Solver.

`static MObject startFrame ;`
The simulation will start from this frame. No computations are made before.

Also, not present in eq.8 but described in You, are the boundary tangent attributes, that describe the angle of incidence between the deformed nodes (nodes) and the boundary regions. They affect the shape even if there is no force present, as shown in figure Fig 6. There are four, one for every boundary :

- Tangency at boundaries attributes

`static MObject su0 ;` Tangent on nodes with u = 0.
`static MObject su1 ;` Tangent on nodes with u = 1.

`static MObject sv0 ;` Tangent on nodes with v = 0.
`static MObject sv1 ;` Tangent on nodes with v = 1.

All these attributes are keyable, visible and have direct effect on the deformation region, but it only will be visible when the elasticNode is updated. And this only happens when the time line changes, it is, the frame changes. By doing so, extra unnecessary calls to the solver are avoided. This has a direct effect on the workflow for the animator: only updates the node when required after tweaking the values searching for a better combination.

b) Mesh Attributes:

Storing and updating mesh data inside the elastic node is heavily defined by the data flow model of Maya, and has to be treated with care in order to avoid loops between this and other nodes that call themselves to update one from another.

- Mesh attributes:

`static MObject inputMesh ;`
Will receive the original mesh (meshOrig) data block to deform that will be copied into inputMeshObject.

`static MObject outputMesh ;`
Receives finalMeshData and sends its block to elasticMesh.

`MObject inputMeshObject ;`
Input for the origin mesh to deform.

`MObject finalMeshData ;`
Data block to operate on.

`MObject solvedMeshData ;`
Stores the solved data from the solver.

The overall flow of mesh data follows the next algorithm:

- Query the inputMesh data.
- If the simulation has not started, copy inputMesh in finalMesh and create solvedMesh from the same data.
- For the first frame of the simulation, gather the data from finalMesh, and call the solver.
- Update solvedMesh
- Once the solver is done, output finalMesh. solvedMesh stores the computed data for next frame.
- Second frame of simulation, copy solvedMesh into finalMesh.
- Gather data from finaMesh and call the solver.
- Update solvedMesh.
- Output finalMesh.
- If current frame is equal or less than startFrame again, reset the scenario: copy inputMesh in finalMesh and solvedMesh.

As seen , the output mesh (finalMesh) is always the mesh data from the previous frame (solution) to the one we are in. With this procedure, no loops are incurred into. This step is crucial for the performance of the node and for the implementation of animation on the node. See fig for a bette understanding of the data flow.

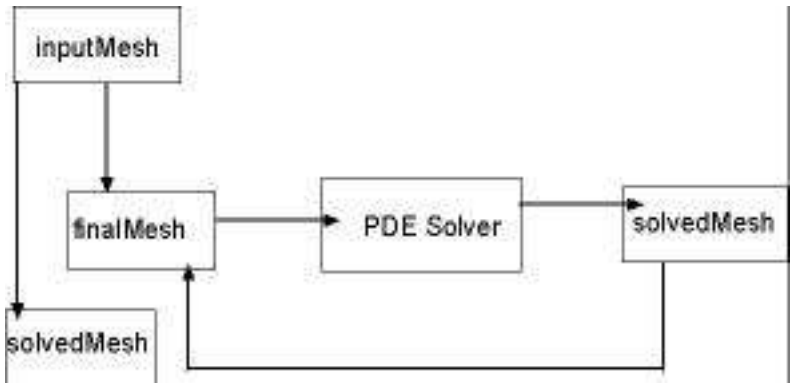


Fig 3. Mesh selector data flow inside the elastic node and its relation with the ODE Solver. First frame of simulation , the input mesh is copied into all the mesh data blocks. When the simulation starts, the solvedMesh data is used as final mesh.

b) Internal Attributes:

This attributes are responsible of parsing, storing and receiving data to and from the solver or other nodes:

- Attributes connected to other nodes:

```
static MObject currentTime ;
```

Time value,receives the current frame from time node inside Maya. Required to decide when to make calls to the solver.

```
static MObject inputForceComponents ;
```
Components of the force vector affecting this shape.

```
static MObject inputForceCenters ;
```
World position of the force. Used to define what nodes are affected and in what region they lie.

```
static MObject deformationRadius ;
```
Radius from within deformation occurs.

```
static MObject forceRadius ;
```
Vertices inside this volume receive force.

```
static MObject forceMag ;
```
Magnitude of the force.

The mesh data is responsible of holding the right data block to operate on. From it, the relevant information required by the solver is extracted:

- Vertex index evaluation

```
MIntArray arrayDeformN ;
```
Index of nodes included on the evaluation.

```
MIntArray arrayBoundaryN ;
```
Index of boundary nodes

```
MIntArray arrayForceN ;
```
Index of nodes under the influence of a force.

```
MIntArray previousDefN ;
```
Index of the nodes from previous PDE evaluation, or state, to decide what velocities need update or are zero (new nodes to deform).

- Arrays to pass the required parameters from Maya to the PDE Solver.

```
MDoubleArray arrayU, arrayV ;
```
U and V coordinates arrays for the deformed vertices or nodes.

- MFloatVectorArray to store the localPositions of the vertices.

```
MDoubleArray arrayPx, arrayPy, arrayPz ;
```
Array storing the local X position of the deformed vertices. This are the arrays that the PDE Solver will update with new positions

- Force components array

```
MDoubleArray    forceComponentsArrayX,
                forceComponentsArrayY,
                forceComponentsArrayZ;
```

Arrays storing the force components of the deformed vertices.

- Velocity attributes

`MDoubleArray Vtx, Vty, Vtz ;`
Components of the initial velocity for the deformed vertices. Also updated in the PDE Solver.

`MDoubleArray solvedVtx, solvedVty, solvedVtz ;`
Components of the solved velocity for the previous state deformed vertices.

5.2.2.- Elastic Force Node:

The main task of the elastic force node is to help the animator define the properties of the force easy and intuitive. The force is considered a tensor, defined by a magnitude and a direction. The deformation region is defined, as the force region, using two volumes with two radius independently modified by the user. This data is sent to the elastic node, which is the one that does the final calculations.

MPxLocator proxy class is used to derive from this node because it has the function draw (), that allows the developer to use OpenGL functions that generate shapes in the viewport. With simple routines, three circles, one for each axis on ech volume, are drawn, with feedback purposes. Also, a vector drawn from the center of the spheres helps to specify the direction of the force.

- Functions:

`static MStatus initialize() ;`
Defines and initializes the attributes and their relationships.

`static void* creator() ;`
Calls the constructor.

`virtual void draw();`
Draws the locator in Maya using OpenGL.

`virtual bool isBounded() const ;`
In order to have bounding box evaluation, this function is set to true.

`virtual MBoundingBox boundingBox() const ;`
Bounding box evaluation function.

`bool getDeformCircle ( MPointArray &pts ) const;`
This functions define the point in the circles of the deformation radius. Are called from the draw function.

With the elasticForceNode the user has a clear feedback about the regions where force and deformation takes place, but the actual vertex evaluation is done in the elasticNode. This values are passed to it through standard Maya´s data flow connections in the dependency graph. The attributes in the elastic force node are:

- Attributes:

`static const MTypeId id ;`
Id of the node inside Maya. Must not clash with exisitng ones.

`static const MString typeName;`
Type name of the node. Defined in the plugIn main.

`static MObject deformationRadius ;`
This attribute defines de volume sphere that will contain the deformed vertices of the elastic shape.

`static MObject forceRadius ;`
This attribute defines de volume sphere that will contain the force vertices of the elastic shape.

`static MObject magnitude ;`
Magnitude of the force vector.

`static MObject forceVector ;`
Vector attribute containing the direction of the force.

5.2.2.- Initializing the plugin:

The initialization and uninitialization of the plug in takes place in the pluginMain.cpp file. It simply defines the actions to take when loading and unloading the plugin, assigns the id to the nodes and the name of the node type if any.

5.3.- Solving the mesh.

In fig the mesh data flow and its relation with the PDE solver is described. Here a more in depth discussion about the compute function and how the data is passed to it is presented. In figure 6 the structure generated in the Hypergraph in Maya helps to visualize the data flow model.

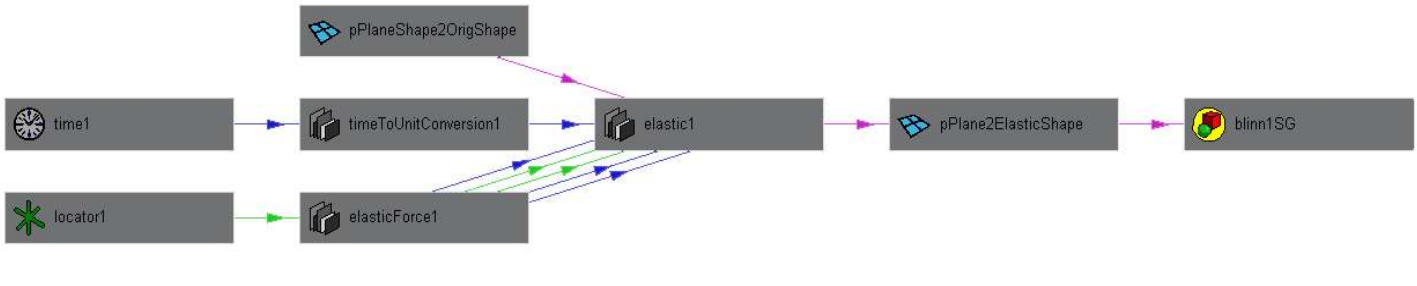


Fig 4.Data flow model in Maya.

- `virtual MStatus compute(const MPlug& plug, MDataBlock& data);`

The first thing the compute function does is create data handlers to store all the attributes that will be required for the evaluation. This attributes come from two sources:

- User input.
- Other nodes.

The force parameters that are connected to the elastic node are stored in Maya data types, are of the second type:

```
MVector vforcesComp = forcesComponentsHnd.asFloatVector() ;
```

And the magnitude:

```
double fmagnitude = forceMagHnd.asDouble() ;
```

For a better and more intuitive force manipulation, it is defined by its direction and magnitude, so using the unit vector in the given direction:

```
MVector forcesVector ;
//! Create unit vector for better user interaction.
double modulus = vforcesComp.length() ;
// Fix division by zero !
if ( vforcesComp.x == 0 || vforcesComp.y == 0 ||vforcesComp.z == 0){
            forcesVector.x = 0 ;
            forcesVector.y = 0 ;
            forcesVector.z = 0 ;
      }else{
            //* X, Y and Z components
            forcesVector.x = fmagnitude*((vforcesComp.x) / modulus) ;
            forcesVector.y = fmagnitude*((vforcesComp.y) / modulus) ;
            forcesVector.z = fmagnitude*((vforcesComp.z) / modulus) ;
            }
```

And the dynamic and elastic properties are from the first type: stored and passed directly to the solver by the user as parameters:

```
// Store the mechanical properties values.
//! Density of the body. Treat it as a double.
double dens = densityHnd.asDouble() ;
//! Damping parameter. Treat it as a double.
double damp = dampingHnd.asDouble() ;
// Young's module and Poisson's ratio related attributes
//! Parameter B
double b = parameterBHnd.asDouble() ;
//! Parameter C
double c = parameterCHnd.asDouble() ;
//! Parameter D
double d = parameterDHnd.asDouble() ;
// Boundary tangents.
//! Tangent at u = 0
```

```
double su_0 = su0Hnd.asDouble() ;
//! Tangent at u = 1
double su_1 = su1Hnd.asDouble() ;
//! Tangent at v = 0
double sv_0 = sv0Hnd.asDouble() ;
//! Tangent at v = 1
double sv_1 = sv1Hnd.asDouble()
```

As mentioned earlier, a key point on the elastic node is to determine the proper mesh data block to act over, depending on if it is updated or not. This is defined in the next selector routine:

```
  //!*******************************
  //! MESH DATA SELECTOR
  //!*******************************
//* Based on the evaluation of the PDE Solver, chooses one or
another mesh data.
//* By default copy the input mesh before simulation starts.
  if (time <= stFrame ){
       //* Create solvedMeshData block
       solvedMeshData = meshDataFn.create() ;
       //* Create finalMeshData block
       finalMeshData = meshDataFn.create() ;
       //*Use the originalMesh as finalMeshData for the first
       evaluation.
       meshFn.copy(inputMeshObject, finalMeshData) ;
       //* And copy it into solvedMeshData as well, so it can be
       updated on the PDE Solver.
       meshFn.copy(inputMeshObject, solvedMeshData) ;
}else{
       //! In case of previous evaluations of the inputMesh, copy
       solvedMeshData in finalMeshData
       meshFn.copy(solvedMeshData, finalMeshData) ;
  }
//* Operate over finalMeshData.
meshFn.setObject(finalMeshData) ;
```

With the data to operate on ready, the first thing to do is define the deformation and force regions and the vertices that are inside. Due to the numerical integration method employed, as mentioned in 4.3, and the equation used from eq.8, the affected vertices are defined based on their (u, v) coordinates. Also a more finer filtering must be done with the objective of determine the boundary vertices and those that reveice forces.

Using the position in world space, every distance to the force center from every vertex is iterated. Using the MItMeshVertex (maya mesh iterator), the regions are defined as follows:

- For every vertex, Find its distance to the force.

o If the distance is less than the deform radius, store the (u,v) coords in uDeformed, vDeformed

o Also, if it is less than the force radius, store the (u, v) coords in uForce vForce

These arrays store the (u, v) coordinates of the affected vertices. With them, a search for the minimun and maximun of the deformation region in parametric space (u,v) is performed to define what vertices should be defined as boundary, deformed or set as points of application of forces. The reason to do it is that is to include on the simulation some vertices that may be away from the force, out of the volume, but that must be affected in order to generate the MxN grid required on the FDM, as stated in 4.3

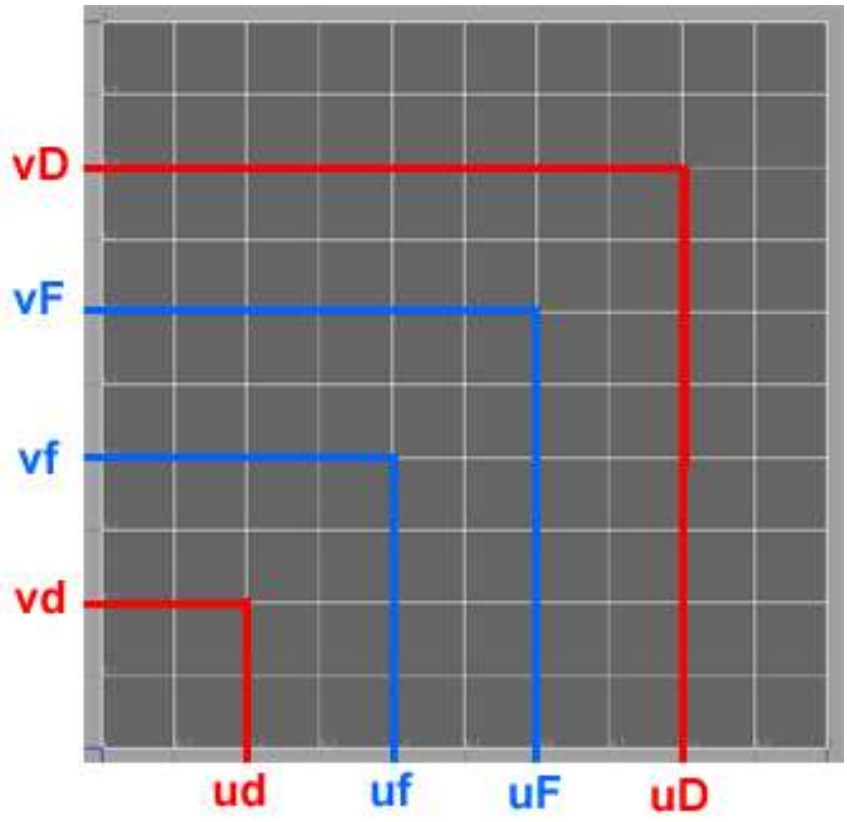


Fig 5. Min and max (u,v) coordinates defining the deformation and force regions.

Min and max (u,v) coordinates statement used to define deformation regions. Same applies for the force region. Note the default values assigned to the uD, ud, vD, vd: they are set out of the bounds of where the shape is defined in order to avoid unexpected deformation regions:

```
//* If there are deformed vertices.
if( uDeformed.length()!=0){
	uD = uDeformed[0] ;
	ud = uDeformed[0] ;
	vD = vDeformed[0] ;
	vd = vDeformed[0] ;
	//* Find the maximun and minimun coordinates.
	for (s = 0 ; s < uDeformed.length() ; s++ )
	{
		if(uDeformed[s] > uD)
			uD = uDeformed[s] ;
		if(uDeformed[s] < ud)
			ud = uDeformed[s] ;
		if(vDeformed[s] > vD)
			vD = vDeformed[s] ;
		if(vDeformed[s] < vd )
			vd = vDeformed[s] ;
	}
}else{
	//! Default values for regions
	uD = -1.0 ;
	ud = -1.0 ;
	vD = -1.0 ;
	vd = -1.0 ;
}
```

Once the intervals are defined, the final iteration along the mesh data is made, and the criteria to define if a particular vertex is affected is not based on its world position: its (u,v) coordinates are used instead:

- For every vertex, get its index and (u, v) coordinates.
- If the coordinates are inside the deform region, store the index in the deformN array, and (u,v) coordinates in arrayU and arrayV.
- And store its local position in Px, Py, Pz arrays.
- Also, if it is inside the force region, store its index in the forceN array and the force components in forceX, Y, Z.
- If not, if it lies on the boundary with u or v equal to uD, ud, vD, vd, store its index in the boundaryN arrays. Append 0.0 on its force arrays to make sure they are same sized.
- If not, it is a deformed vertex that do not receive force, so append 0.0 on its force arrays.
- Go to next vertex.

In order to define the number of loops inside the PDE Solver, the total number of nodes on both U and V directions must be defined:

- Loop through all the U values from arrayU
- When the value equals the first position in the array, that means one side has been covered.
- The number of loops is the number of nodes along V.

And for U:

- Loop through all the V values from arrayV
- When the value differs from the first position of the array, that means the side has been covered.
- The number of loops is the total number of nodes along U.

The last data that the PDE solver needs are the velocities of the vertices involved on the simulation. It is zero if it was not computed previously. If solved, the velocity should be restored from the previous deformed state, and resent to the solver in its right position of the array.

By saving in solvedVtx, solvedVty and solvedVtz inside the solved the velocities from the previous evaluation, and in previousDefN the indexes of the previous affected or displaced vertices, a comparison between the two arrays can be made, and a new Vtx, Vty and Vtz array is created, containing the pertinent velocities:

- 0 ; if the node is newly deformed.
- solvedVtx[i], solvedVty[i], solvedVtz[i] ; if was previously affected.

The last routine, once data is ready, is the actual call to the PDE Solver. Solving the equations is a time consuming task, so the solver should not be called unless it is really required. Two conditions are checked before calling it:

- Start frame: If time is less than starting frame, no call to the solver is made.
- Number of nodes to deform: If zero, no call is made.

After the PDE call, the solvedMesh data is updated and ready for the next evaluation , so the finalMesh can be sent to the output plug.

- `void PDESolver();`

This function has 4 tasks:

1.- Clear the arrays of solved data and copy the deformed nodes indexes into a new previousDefN array that will be used to compare on the next solving with the objective of defininig the velocities on next iteration. It is key to understand what data and from what state each object contains, in order to define the proper evaluations.

```
//* Clear previous arrays.
previousDefN.clear() ;
solvedVtx.clear() ;
solvedVty.clear() ;
solvedVtz.clear() ;

//! Copy in  previousDefN the indexes of deformN to compare deformed
velocities for the next evaluation
previousDefN = arrayDeformN ;
```

2.- Copy all the maya data types into C++ arrays that the eqSolver can use to generate the finite difference equations. The method MDoubleArray::get() is used to copy into the c_arrayPx the contents of arrayPx, for example. Same applies with the other required arrays:

```
//!** Positions **
double *c_arrayPx = new double [numberDeformed] ;
       arrayPx.get(c_arrayPx) ;
double *c_arrayPy = new double [numberDeformed] ;
       arrayPy.get(c_arrayPy) ;
double *c_arrayPz = new double [numberDeformed] ;
       arrayPz.get(c_arrayPz) ;
```

3.- Calls the EqSolver and keeps updating it until totalTime == 1/24, when the updated c_arrayPx, c_arrayPy and c_arrayPz are updated with the new positions and c_Vtx, c_Vty and c_Vtz with the new velocities:

```
L10: tTime += dTime;
       //*Call EqSolver to generate finite difference equations and solve them.
       EqSolver(b,c,d,dens,damp,su0,su1,sv0,sv1,h,c_arrayPx,totalU,  totalV,  TotalNode,
       Neq, dTime,c_forceComponentsArrayX,c_Vtx);
       EqSolver(b,c,d,dens,damp,su0,su1,sv0,sv1,h,c_arrayPy,totalU,  totalV,  TotalNode,
       Neq, dTime,c_forceComponentsArrayY,c_Vty);
       EqSolver(b,c,d,dens,damp,su0,su1,sv0,sv1,h,c_arrayPz,totalU,  totalV,  TotalNode,
       Neq, dTime,c_forceComponentsArrayZ,c_Vtz);
       //* Output the calculated results when reaching the specified time.
if(tTime < 1.0/24.0) goto L10;
```

4.- Updates the new positions and velocities on the Maya arrays:

For new positions, solvedMesh is updated:

```
//! For the vertices in solvedMesh
for (  j = 0 ; j < iterMesh.count() ; j++ )
{
	//! Get the index
	int index = iterMesh.index() ;
	//! Create a MPoint to the particular vertex being evaluated to
	store its positions
	MPoint &p = vertArray[j] ;

	//! For all the deformed vertex index
	for (  v = 0 ; v < arrayDeformN.length() ; v++ )
	{
		//! If the index from mesh is on the deformed array
		if ( index == arrayDeformN[v])
		{
			//! Update the local positions Px, Py, Pzfrom the
solver
			p.x = c_arrayPx[v] ;
			p.y = c_arrayPy[v] ;
			p.z = c_arrayPz[v] ;
		}
	}
//! Go to next vertex.
iterMesh.next() ;
}
//! Once updated, the new positions are passed to solvedMeshData
meshFn.setPoints(vertArray) ;
meshFn.updateSurface() ;
```

And the new velocities:

```
for ( v = 0 ; v < numberDeformed ; v++)
{
	// And update velocities.
	solvedVtx.append(c_Vtx[v]) ;
	solvedVty.append(c_Vty[v]) ;
	solvedVtz.append(c_Vtz[v]) ;
}
```

This method makes solvedMesh to be ready for the next frame, not for current. On current frame the previous frame solved state is output. This make the node control the data and the way it is updated.

- `void EqSolver() ;`

This function is the equation generator: generates the linear matrix equation from the parameters for X, Y, and Z, applies boundary conditions, etc. that will be solved using standard techquines in ciggj() ;

It is important to mention that the minimun size of the grid defined by the deformation region must be at least 3X3 with one force vertex. Otherwise, the FDM is not able to calculate the new positions, because there is not data enough.

- `int ciggj() ;`

This is a standard technique for solving linear equations that can be found in [ ].

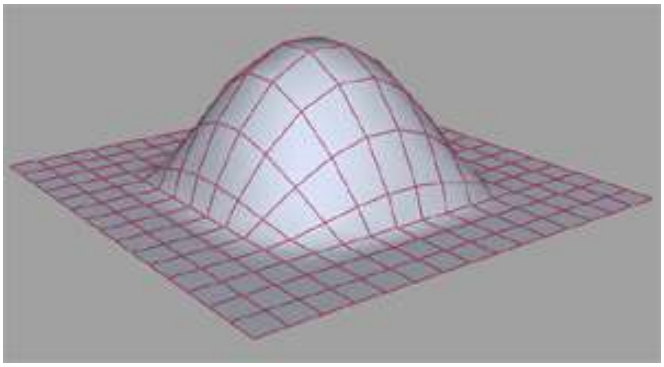
A last mention about the elasticForceNode must be considered. Elastic force has direct relation on the deform regions: it gives the animator the desired feedback to visualize where the shaoe will be affected and how. That is why its functions are more related with the human interface than about performing any computation or produce results. It acts as a pure translator between the elastic node and the animator.

The function that mostly helps to this task is the draw(); function. It uses OpenGL routines inside Maya. That is how the circles are drawn. An important point on this function must be stated: it must leave OpenGL in exactly the same state it was before the function was called. glPushAttrib() and glPopAttrib() help to this point.

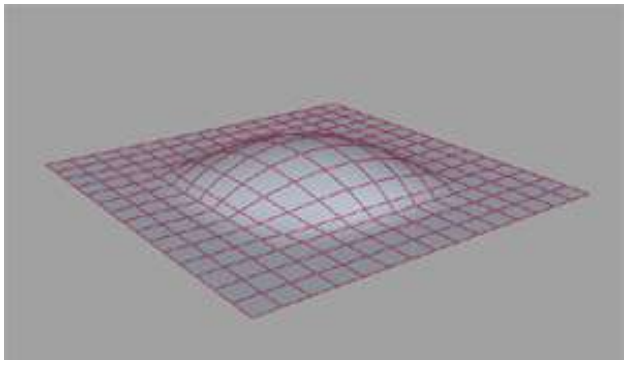
Also, a simple bounding box routine is implemented, as a form of a 20X20 cube from (-10, -10, -10) to (10, 10, 10) in order to have fully operational the zoom function inside Maya.

6.- Simulation examples.

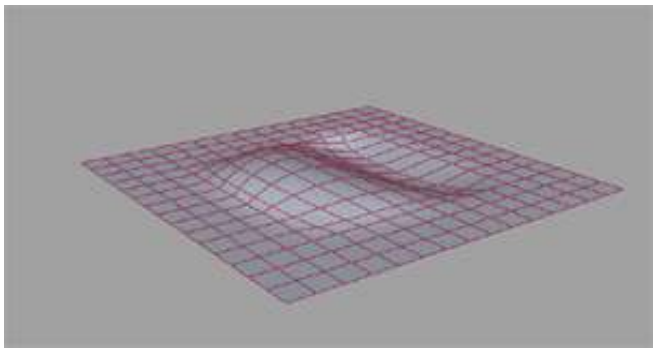
Some interesting results have been achieved by tweaking the parameters. In figure 7, the effects of a simple force applied to the plane defines a deformation shape, that has different aspect when modifying the elastic properties or the tangent parameters.



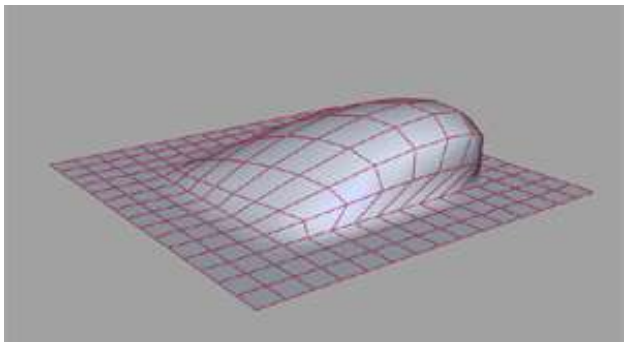
Only force applied.



Effects of increasing the density parameter.



Tangent at boundary u = 0 set to 30.

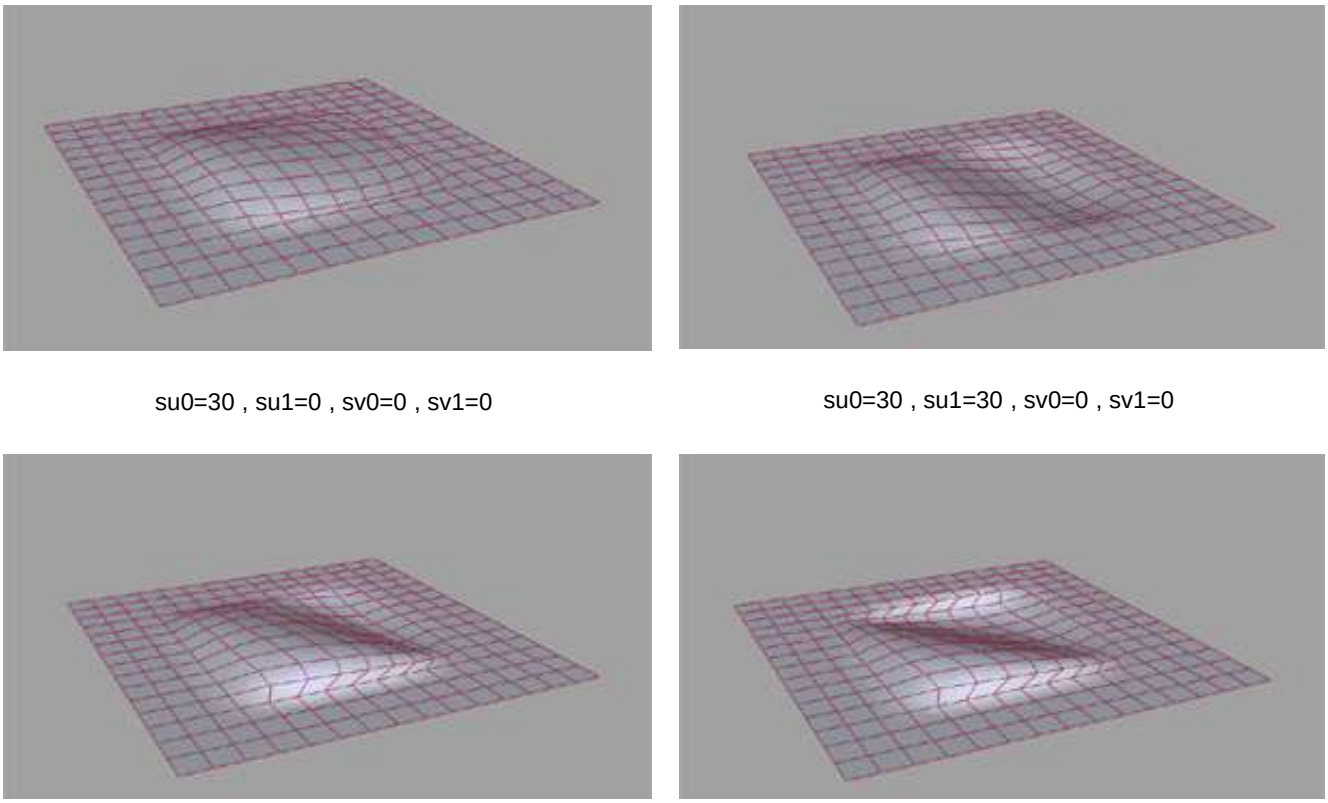


Tangent at boundary v = 0 set to 30.

Fig 8. Simple force on the Y axis applied to a shape and effects of the combined parameters. t = 10 .

The effects of density and damping changes are only visible during animation. They are powerful parameters that can define nice behaviours, from a more fluid like to a more rigid or stiff body. Increasing the density of a solid, the inertia is being increased, with the result of a more resistence to change the state: it will need a bigger force to affect the surface, but once it is done, the force to stop it will be big as well. Animating this parameter has a direct effect on the material look. If the damping attribute is increased, the body shows a bigger force due to the velocity, that will make it feel like as more elastic.

The tangent values at the boundaries (fig 9) have as a result a deformed shape even if there is not force applied. When combined with force, the animator has direct control over the deformed shape. This attributes can describe shapes even if there are no forces present on the system.



su0=30 , su1=0 , sv0=0 , sv1=0

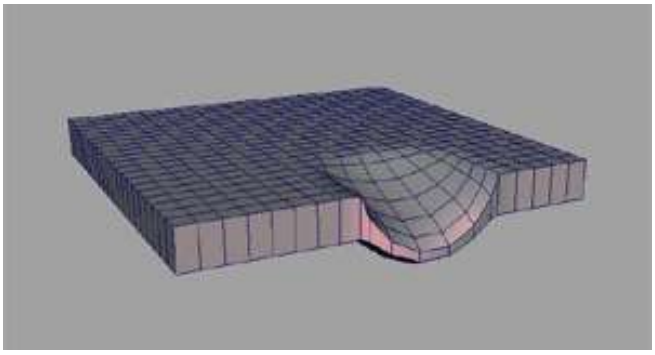
su0=30 , su1=30 , sv0=0 , sv1=0

su0=30 , su1=30 , sv0=30 , sv1=0

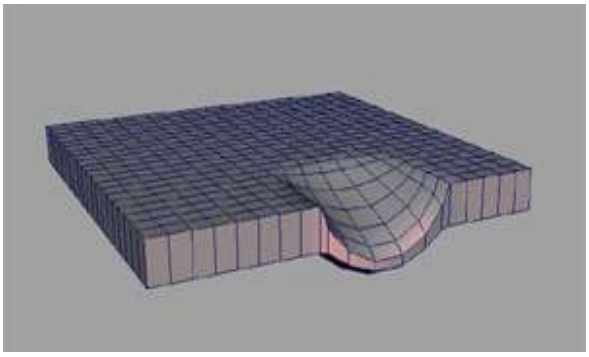
su0=30 , su1=30 , sv0=30 , sv1=30

Fig 9. Effects of increasing the tangent boundary parameters with no force. t = 1.

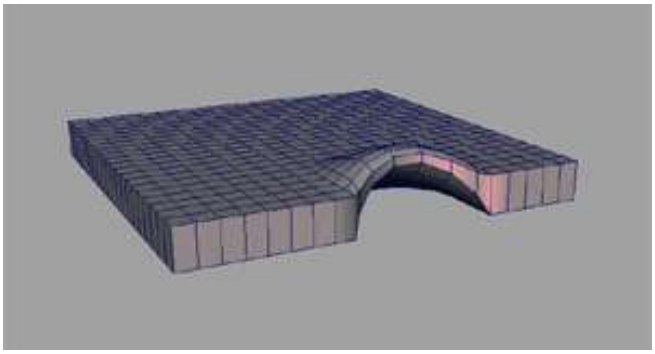
Physical behaviours can be described fast and efficiently, like the one in figure 10, where a rubber like behaviour is displayed over a look-like metal piece. Notice the different deformation for same frame by just changin the deltaTime parameter that controls the time step of the simulation. But note that a bigger time step means a less accurate solution as well. Which one is the correct solution?



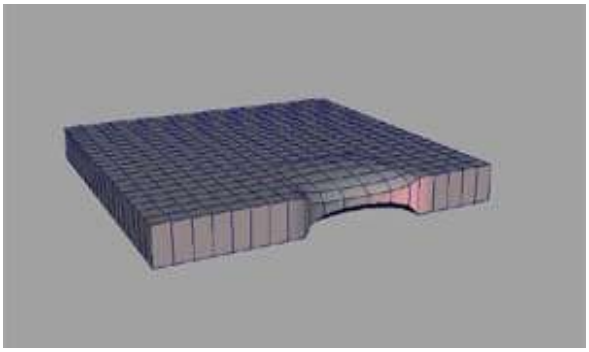
t=25 ; dt=0.01; Fmag = 300.



t=25 ; dt=0.01; Fmag = 300.



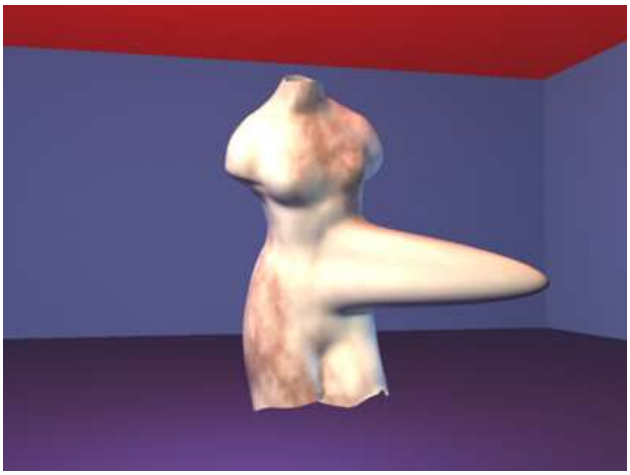
t=30 ; dt=0.01; Fmag = 0.



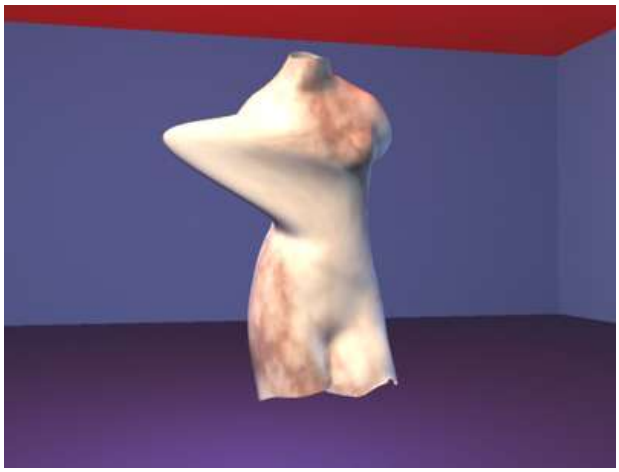
t=30 ; dt=1; Fmag = 0.

Fig 10. Different solutions depending on the time step taken. The left images took less time to calculata, but notice the right shows more detailed and accurate shape, specially upon removal of the force, at frame 30.

Moving a force throgh space is also posible, as show in the deformed venus (fig 11).



Deformation radius = 5
Force radius = 10



Deformation radius = 10
Force radius = 5

Fig 11. The same force is displaced to another location, producing different deformations on the mesh.

At last, varying the force direction offers more levels of control. Notice the deformation radius defined by the dark sphere volume and the force volume by the light one. Also note the squared deform region

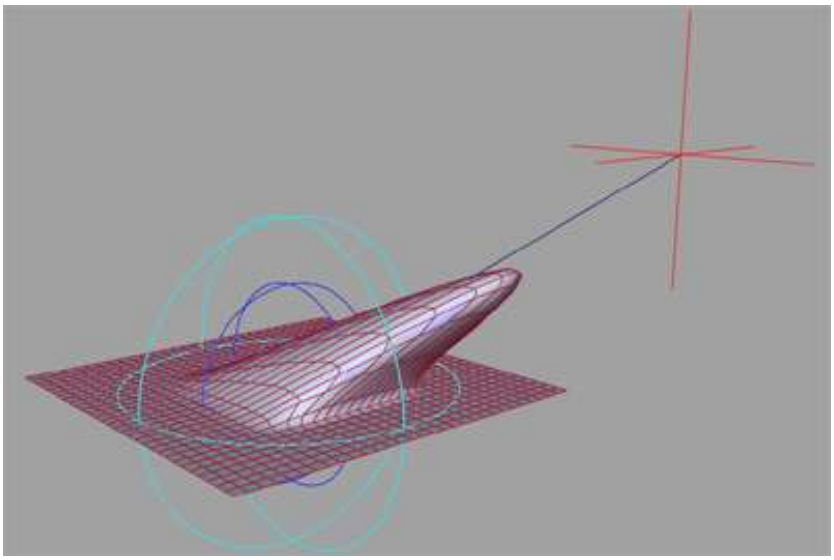


Fig12. Force node offers more levels of control and is intuitive.

7.- Conclusions and future work.

The development of the tool implementing elastic behaviour in Maya is a success. The equation proposed in You is seamless integrated with Maya, and can be used in combination with all the other existing tools present in the software. Interesting behaviours have been achieved and the tool proved to be fast, efficient and robust. Almost every shape can be turned into elastic and deformed, without losing control over the original data, due to the data flow model followed.

Because the finite differenc method is used to solve the equation, only polygonal meshes with even distribution of their (u,v) coordinates along a grid and a correct indexing of their vertices can be deformed. But developing the right algorithms to filter a mesh, this restriction can be overriden.

But from it, the hardest to solve is the (u,v) grid. In production almost never the (u,v) grid is displayed as the finite difference method requires. The layout is more often irregular and most of the time some vertices use more than one texture coordinates. This issue can be avoided if another numerical integration method is used, as the weighted residual method, that can deal with uneven grids, or in the other hand, with algorithms to create a virtual parametric space for the affected nodes could be developed, to make the data generation for the equation independent from the (u,v) texture coordinates used in Maya. It would be easy to project only the deformed region into a specific grid and solve the equation. It seems better to emply other method, however, because not always the shapes can be projected into even grids, because the number of on every direction changes, or the mesh is not regular.

Although a set of patches can be used to define a model, this is a time consuming task that would not be accepted in production pipelines. Another solution would be to create a proxy model of the deformed area, solve it and transfer the result to the original mesh. Those are techniques widely used in the industry, but rely more on the skills of the animator than on the development of the tool.

On the side of the vertex indexing, renumbering of the mesh would not be needed. A proper mapping technique from the vertices into a that virtual parametric space would make the method totally independent of the way maya treats polygons. Only the object positions and initial velocities would be required. With the implementation of those methods, also NURBS and subdivision surfaces can be turned into elastic and deformed in consequence.

Currently only one force is supported on each elastic body. But implementing more than one is easy, and can be achieved through two ways: definition of a deformation region and forces inside it, that would lead to a new elasticForce type (would be called multipleForce), or connecting more than one force to the node, by implementing array attributes. Both methods are compatible and should be the next step. Also different types of forces could be implemented, as radial, line forces, etc...The resolution of the surface would involve one call to the PDE Solver at a time per force.

Because the method is subkect to boundary conditions, alternative method to describe global deformations would be interesting. By doing so, more interesting behaviours can be achieved, like cloth or moving deformable bodies.

8.- References:

[1] R.Parent “A System for Generating Three-Dimensional Data for Computer Graphics.” Ph.D. dissertation. Ohio State University, 1977.

[2] A.Barr “Global and Local Deformations of Solid Primitives.” Proceedings of SIGGRAPH 1984, Computer Graphics Volume 18, Number 3, pp 21 -30.

[3] T. Sederberg,S.Parry, "Free-form deformation of solid geometric models," Computer Graphics, 20, 4, 1986, (Proc. SIGGRAPH), 151-160.

[4] W.Press, B.Flannery, S.Teulkosky, W. Vetterling “Numerical Recipes” Cambridge University Press, Cambridge 1986.

[5] R.Barzel, A.Barr "Modeling with Dynamic Constraints," in Topics in Physically Based Modeling, SIGGRAPH 1987. Tutorial 17 Notes.

[6] D.Terzopoulos, J. Platt, A. Barr, K.Fleischert “Elastically Deformable Models.” Proceedings of SIGGRAPH 1987 Computer Graphics, Volume 21, Number 4, pp 205 – 214.

[7] A. Witkin, K. Fleischer, A.Barr “Energy Constraints in Parameterized Models” Proceedings of SIGGRAPH 1987 Computer Graphics, Volume 21, Number 4, pp 225 – 232.
[8] J.Platt, A.Barr “Constraint Methods for Flexible Models.” Proceedings of SIGGRAPH 1988, Computer Graphics, Volume 22, Number 4, pp 279 – 288.

[9] D.Terzopoulos, J.Platt “Physically based Models with Rigid and Deformable Components.” IEEE Computer Graphics and Applications. November 1988. pp 41 – 51.

[10] D.Terzopoulos, K. Fleiseher. “Modeling Inelastic Deformation: Viscoelasticity, Plasticity, Fracture” Proceedings of SIGGRAPH 1988, Computer Graphics, Volume 22, Number 4, pp 269 – 278.

[11] D.Terzopoulos, J. Platt. “Physically-Based Modeling : Past, Present, and Future”, SIGGRAPH '89 Panel Proceedings. pp 191 – 209.

[12] A. Garcia, A.Lopez, G.Rodriguez, S. Romero, A. de la Villa “Calculo II, Teoria y problemas de funciones de varias variables” Clagsa, Madrid. 1996. ISBN 84-921847-0-1

[13] D. Baraff, A.Witkin “Large Steps in Cloth Simulation” SIGGRAPH ’98

[14] R.Parent “Computer Animation Algortihms and Techniques.” Morgan-Kaufmann 2002, San Francisco. ISBN 1-55860-579-7

[15] D.Gould, “Complete Maya Programming, An extensive guide to MEL and the C++ API.”, Morgan-Kaufmann 2003, San Francisco. ISBN 1-55860-835-4.

[16] H.M Deitel, P.J. Deitel, “C++ How to program.”Prentice Hall 2003, New Jersey. ISBN 0-13-111881-1

[17] R.Barzel “Physically-based modeling for computer graphics : a structured approach”. Boston ; London : Academic Press, 1992

[18] D. G. Zill “A First Course in Differential Equations with Modeling Applications.” Sixth edition.1995. ISBN 0-534-95574-6

[19] L. You, J, J, Zhang. “Fast Generation of 3-D Deformable Moving Surfaces”. IEEE Transactions on Systems, Man and Cybernetics. Vol 33, No 4, August 2003.

[20] Witkin, A., Baraff, D., 2001b. “Physically Based Modeling: Differential Equation Basics.” [online]. California, Pixar. Available from: http://www.pixar.com/companyinfo/research/pbm2001/notesc.pdf [Accessed 5 July 2005].

[21] Baraff, D., 2001b. “ Physically Based Modeling: Implicit Methods” [online]. California, Pixar. Available from: http://www.pixar.com/companyinfo/research/pbm2001/notesd.pdf [Accessed 5 July 2005].

[22] Alias Maya Developer Help. Alias. Toronto.2005

[23] Jian J Zhang, L. You, P. Comninos. “Computer Simulation of Flexible Fabrics” Bournemouth University.

[24] J. Vince, “Geometry for Computer Graphics. Formulae, examples and proofs.” Bournemouth, UK, 2005. ISBN 1-85233-834-2

[25] E. W. Weisstein http://mathworld.wolfram.com/EulerForwardMethod.html Accessed 1/06/2005

[26] E. W. Weisstein http://mathworld.wolfram.com/Runge-KuttaMethod.html Accessed 2/05/2005

[27] E. W. Weisstein http://mathworld.wolfram.com/FiniteDifference.html Accessed 2/05/2005

[28] E. W. Weisstein http://mathworld.wolfram.com/CentralDifference.html Accessed 2/05/2005