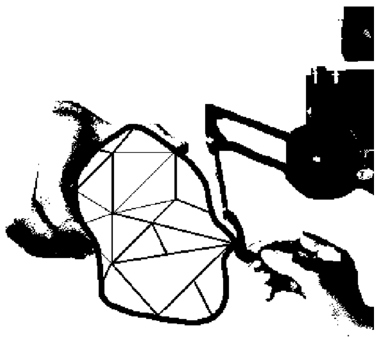
Force-Reflecting Deformable Objects for Virtual Environments

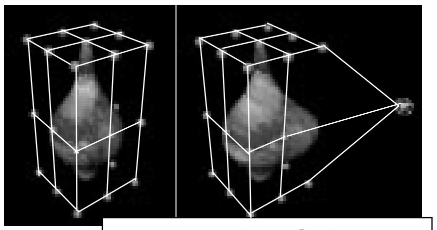


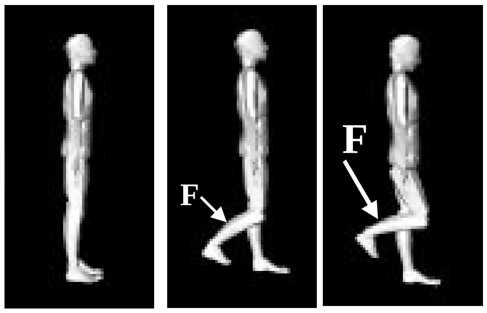
Cagatay Basdogan, Ph.D.
Research Laboratory of Electronics
Room: 36-758, MIT, 02139
basdogan@mit.edu

Graphical display of deformable objects has been extensively studied in computer graphics. With the addition of haptic feedback, deformable objects gain a new characteristic. Now, our models should not only estimate the direction and the amount of deformation of each node but also the magnitude and direction of interaction forces that will be reflected to the user via a haptic device. This tutorial note discusses the modeling and programming principles of force-reflecting deformable objects.

Applications

Surgical Simulation


Haptic Sculpting



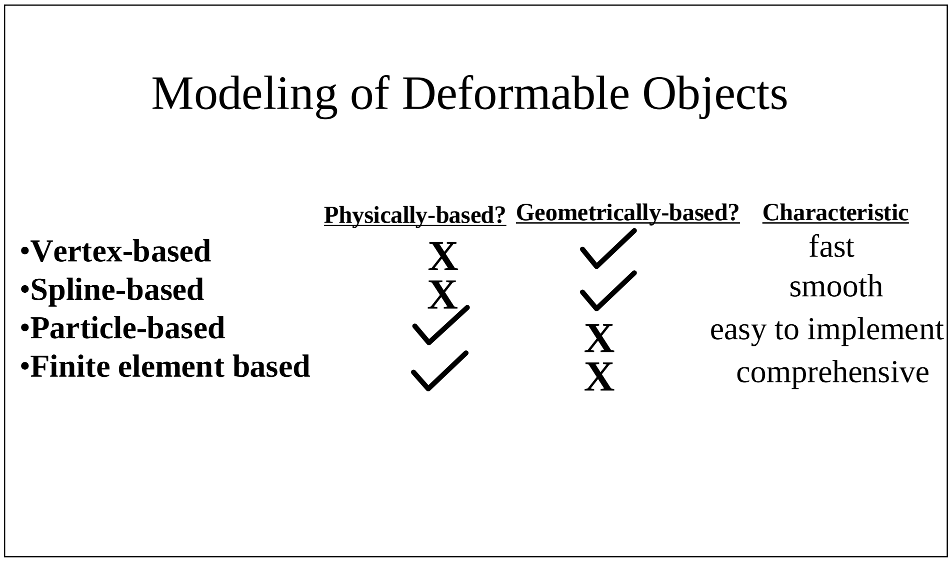
Physically-based Character Animation

Applications:

• Surgical simulators are currently being developed at many research centers and companies to train doctors and residents with new surgical devices and techniques. Conveying to the surgeon the touch and force sensations with the use of haptic interfaces is an important component of a simulator. Force-reflecting deformable models in various fidelities need to be developed to simulate the behavior of soft tissues when they are manipulated with surgical instruments. The developed algorithms should deal directly with geometry of anatomical organs, surface and compliance characteristics of tissues, and the estimation of appropriate reaction forces to convey to the user a feeling of touch and force sensations.
• 3D modeling of deformable objects using NURBS or FFD are well known concepts in CAD. With the addition of force feedback, the interactions will be more intuitive and physically based. For example, various constraints can be implemented naturally using force feedback.
• An animator can intuitively deform the body parts of a 3D character using a haptic device. For example, an animator can use force cues to decide on how much the knee of a 3D character should be flexed at each time frame to make its locomotion more realistic.
• Mechanistic interactions between the melted materials and the manufacturing tools can be studied in virtual environments. For example, an extrusion process can be simulated to better understand the behavior of materials under certain external loads.

Desired properties of force-reflecting deformable models

- reflect stable forces
- display smooth deformations
- handle various boundary conditions and constraints
- display “physically-based” behavior



Modeling of Deformable Objects

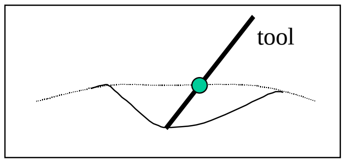
	Physically-based?	**Geometrically-based?**	**Characteristic**
•**Vertex-based**	X	✓	fast
•**Spline-based**	X	✓	smooth
•**Particle-based**	✓	X	easy to implement
•**Finite element based**	✓	X	comprehensive

One way to categorize the deformation techniques is according to the approach followed by the researchers to deform the surfaces: *geometric* or *physically-based* deformations. In geometric deformations, the object or the surrounding space is deformed based purely on geometric manipulations. In general, the user manipulates vertices or control points that surround the 3D object to modify the shape of the object. On the other hand, physically-based deformation techniques aim to model the physics involved in the motion and dynamics of interactions. Models simulate physical behavior of objects under the affect of external and internal forces. Geometric-based deformation techniques are faster, and are relatively easier to implement. But they do not simulate the underlying mechanics of deformations. Hence, the emphasis is on visual display and the goal is to make deformations appear smoother to the end-user. Sophisticated physically based models, although necessary for simulating the dynamics of realistic interactions, are not suitable for fully interactive, real-time simulation of multiple objects in virtual environments due to the current limitations in computational power.

Modeling of Deformable Objects

Vertex-based :

$$\text{Depth} = a_0 + a_2\,(\text{Radial Distance})^2$$

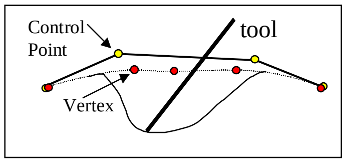


Spline-based :

$$Q(u,v,w) = \sum_i \sum_j \sum_k P_{ijk}\, B_i(u)\, B_j(v)\, B_k(w)$$

$$\Delta P = (B^T B)^{-1} B^T \Delta Q$$

$$Q_{new} = B\,(P + \Delta P)$$

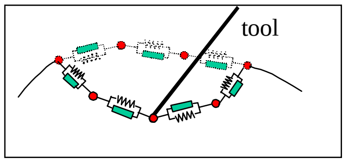


Particle-based :

$$F = ma$$

- F_{spring}
- $F_{damping}$
- $F_{gravity}$

$$\boxed{\begin{aligned} a(t+\Delta t) &= F/m \\ v(t+\Delta t) &= v(t) + \Delta t\, a(t+\Delta t) \\ p(t+\Delta t) &= p(t) + \Delta t\, v(t+\Delta t) \end{aligned}}$$



Geometric and *physically-based* deformation techniques can be sub-grouped as follows:

A. Geometric-based Deformation Models
   - *Vertex-based:* The vertices of the object are manipulated to display the visual deformations.
   - *Spline-based:* Instead of directly transforming the vertices of the object, control points are assigned to a group of vertices and are manipulated to achieve smoother deformations.

B. Physically-based Deformation Models
   - *Particle-based:* Particle systems consists of a set of point masses, connected to each other through a network of springs and dampers, moving under the influence of internal and external forces. In this model, each particle is represented by its own mass, position, velocity, and acceleration.
   - *Finite Element based:* The volume occupied by the object is divided into finite elements, properties of each element is formulated and the elements are assembled together to study the deformation states for the given loads.

For geomeric-based models, we assume that the user will define his/her own force interaction model. The force model will depend on the deformation model. For example, a set of linear/nonlinear springs can be considered between the home and deformed positions of nodes to compute the direction and magnitude of the force vector that will be reflected to the user. In physically-based modeling, the model automatically computes the magnitude and direction of forces applied to each node.

- *Vertex-based*: A region of the object surface in the close vicinity of the collision point (or the nearest surface point) can be locally deformed. In order to deform object, we translate all of the vertices within a certain distance (called the *radius of influence*) of the collision point, along the direction of the haptic stylus. For example, the magnitude of translation can be determined using a simple second order polynomial. The degree and the coefficients of the polynomial define the shape of the deformations. For example, if a second degree polynomial with no linear deformation term is assumed ($a_1 = 0$), then the deformation function takes the following form

$$\text{Depth} = a_0 + a_2 (Radial\ Distance)^2$$

where, $a_0 = \text{AP}$ and $a_2 = -\text{AP} / (radius\ of\ influence)^2$. The vector AP is constructed from the coordinates of the stylus tip and the contact point. The *radial distance* is the distance of each neighboring vertex, within the radius of influence, to the collision point.

- *Spline-Based:* Sederberg and Parry (1986) suggested a free-form deformation (FFD) technique for deforming the space that encloses the object. FFD enables the user to interactively modify the object shape by repositioning the lattice of control points that surround the 3D object. Any point within the lattice is defined as:

$$Q(u,v,w) = \sum_{i=0}^{3} \sum_{j=0}^{3} \sum_{k=0}^{3} P_{ijk} B_i(u) B_j(v) B_k(w)$$

or, in matrix form

$$Q = BP \qquad (*)$$

where, P_{ijk} are the control points, and $B_i(u),\ B_j(v),\ B_k(w)$ are known as the third degree Bernstein polynomials or Bezier basis functions. Hsu et al. (1992) suggested a method for direct manipulation of free-form surfaces. In this method, control points are moved such that the resulting surface smoothly reaches its intended position by means of a least squares solution. Assume that a single point of the 3D object is translated an amount of ΔQ and moved to a new location ($Q + \Delta Q$), then Eq. (*) can be rewritten in the following form:

$$(Q + \Delta Q)_{1\times 3} = B_{1\times 64} (P + \Delta P)_{64\times 3} \qquad (**)$$

where, ΔQ and ΔP represent the changes in the position of object point and the control points (recall from Eq. (5) that there are 64 control points). Eq. (**) reduces to:

$$\Delta Q_{1\times 3} = B_{1\times 64}\, \Delta P_{64\times 3}$$

Now, the goal is to calculate the change in the control points for a given ΔQ. This can be achieved through the use of pseudoinverse solution:

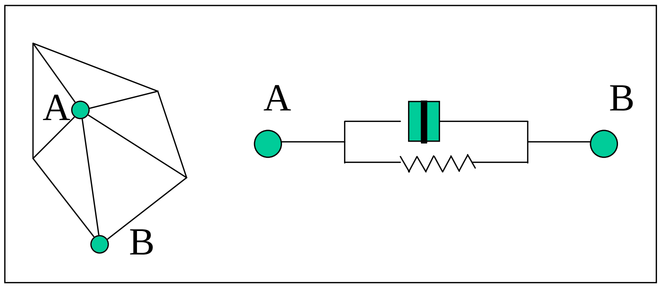
$$\Delta P = (B^T B)^{-1}\, B^T \Delta Q$$

Once the changes in the positions of control points are known, the deformed positions of the object can be calculated from $Q_{new} = B(P + \Delta P)$.

<u>Suggested References:</u>

1. Basdogan C., Ho, C., Srinivasan, M.A., Small, S., Dawson, S., 1998, "Force interactions in laparoscopic simulations: haptic rendering of soft tissues" *Medicine Meets Virtual Reality (MMVR'6) Conference*, pp. 385-391, San Diego, CA, January 19-22.
2. Dachille, F., Qin, H., Kaufman, A., El-sana J., 1999, "Haptic sculpting of dynamic surfaces", *submitted to the 1999 Symposium on Interactive 3D Graphics.*
3. Hsu W.M. Hughes J.F., Kaufman H., 1992, "Direct Manipulation of Free-Form Deformations", *Computer Graphics (Proceedings of the SIGGRAPH)*, Vol. 26, No.2, pp. 177-184.
4. Edwards, J., Luecke, G., 1996, "Physically based models for use in a force feedback virtual environment", Japan/USA Symposium on Flexible Automation, ASME 1996, pp. 221-228.

• *Particle-Based:* Particle systems (also known as mass-spring models) consists of a set of point masses, connected to each other through a network of springs and dampers, moving under the influence of internal and external forces (see figure below).



Each vertex (i.e. node) of the 3D object has a mass and is connected to its neighbors with springs and dampers, moving under the influence of internal and external forces.

The total force applied on each particle can decomposed into *spring, gravitational,* and *dissipative forces.*

$$F_{total} = \left\{ \begin{matrix} F_{spring} = \sum k(l - l_0) \\ F_{gravitational} = mg \\ F_{dissipative} = -bv_{i,j} \end{matrix} \right\}_{i^{th} particle} \qquad (***)$$

Then, the acceleration, velocity, and position of each particle can be updated using the Euler integration method.

$$\begin{aligned} a_{t+\Delta t} &= F_{total} / m \\ v_{t+\Delta t} &= v_t + \Delta t \; a_{t+\Delta t} \qquad (****) \\ p_{t+\Delta t} &= p_t + \Delta t \; v_{t+\Delta t} \end{aligned}$$

Particle systems have been extensively used in computer graphics to simulate the behavior of clothes and fluid flow. This technique is simple to implement since the developer does not need to construct the equations of motion explicitly. Moreover, it is physically-based since it can model the viscoelastic behavior of deformable objects.

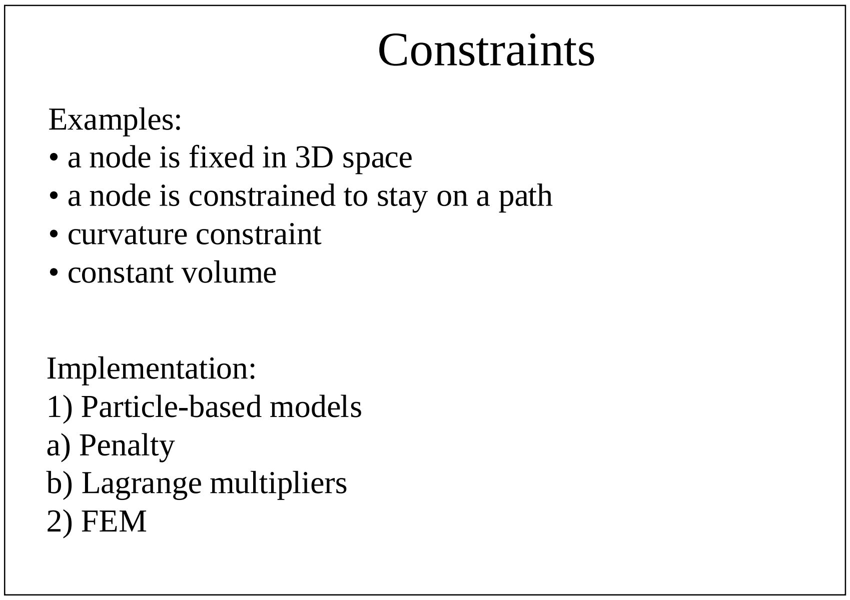
Suggested References:

1. Cover S.A., Ezquerra N.F., O'Brien J., Rowe R., Gadacz T., Palm E., 1993, "Interactively Deformable Models for Surgery Simulation", *IEEE Computer Graphic and Applications*, November, pp. 65-78.
2. Ng, H., Grimsdale, R., 1996, "Computer Graphics Techniques for Modeling Cloth", *IEEE Computer Graphic and Applications*, September, pp. 28-41.
3. Joukhadar A., Laugier, C., 1995, "Fast Dynamic Simulation of Rigid and Deformable Objects", IEEE/ICRA, pp. 305-310.
4. Witkin A., Barraff D., Kass M., Tutorial notes on " *An Introduction to Physically-Based Modeling*", SIGGRAPH'98.
5. Lee, Y., Terzopoulos, D., Waters, K., 1995, "Realistic Modeling for Facial Animation", *(Proceedings of the SIGGRAPH),* pp. 55-62.
6. N. Swarup, "Haptic Interaction with Deformable Objects Using Real-Time Dynamic Simulation", M.S. Thesis, Mechanical Engineering Department, Massachusetts Institute of Technology, (1995).

- *FEM-Based:* The finite element models come in various forms and the selected model depends on the type of loading, element, and shape functions. We do not present a model in here due to the limited space, but the following references are quite helpful in developing finite element models for simulating deformable objects.

Suggested References:

1. Rao, S. S., 1988, “The finite element method in engineering”, Pergamon Press, NY.
2. Zeinkiewicz, O.C., 1979, “The finite lement method”, McGraw-Hill, New Delhi.
3. Bathe, K., 1996, “Finite Element Procedures”, Prentice Hall, New Jersey.
4. Bro-Nielsen, M., Cotin, S., “Real-time Volumetric Deformable Models for Surgery Simulation using Finite Lements and Condensation”, EUROGRAPHICS’96, Vol. 15, No. 3, pp. 57-66.



Constraints

Examples:
• a node is fixed in 3D space
• a node is constrained to stay on a path
• curvature constraint
• constant volume

Implementation:
1) Particle-based models
a) Penalty
b) Lagrange multipliers
2) FEM

Constraints:
So far, we have discussed various modeling techniques for simulating force-reflecting deformable objects. To control the deformations and to make the simulations more realistic, constraints have to be implemented into models. Several types of contraints can be mentioned:

- a node is fixed in 3D space
- a group of nodes has to follow a path
- curvature constraints has to be specified for modifying free form surfaces
- volume of the object has to be kept constant

1) *Implementation of constraints to particle-based models:*

Many techniques have been suggested to implement constraints. We briefly discuss only two of them in here: (a) penalty methods and (b) Lagrange multipliers (Interested readers may find the details of these techniques in the suggested references). In general, the constrained force estimated through (a) penalty or (b) Lagrange multiplier technique is added to the unconstrained force computed through equation (***). Then, the total force ($F_{total} = F_{constrained} + F_{unconstrained}$) is inserted into Eq. (****) to update the accelaration, velocity and position of each particle.

a) Penalty methods

Calculate the constrained force using the following formulation:

$$F^{constrained}{}_{i} = (-k_s G - k_d \dot{G}) J$$

where, G(u) is your constraint function that has to be satisfied, u reperesents your nodal displacements, k_s and k_d are spring and damping coefficients that can be adjusted to satisfy constraints, and J is jacobian ($J = \partial G / \partial u_i$). For example, if we want to fix a node (i^{th} node) in 3D space, we define G(u) as G(u) =u_i. Then, $F^{constrained}{}_{i} = (-k_s u_i - k_d \dot{u}_i)$.

b) Lagrange multipliers

First, solve the following equation for λ and then insert the solution into $F_{constrained} = j^T \lambda$ to estimate the constrained force.

$$JM^{-1} J^T \lambda = -\dot{J}\dot{u} - JM^{-1} F_{unconstrained} - k_s G - k_d \dot{G}$$

where, M is the diagonal mass matrix, λ is a vector of Lagrange multipliers.

Suggested Readings:

1. Witkin, A., 1997, “An Introduction to Physically Based Modeling: Constrained Dynamics, SIGGRAPH’97 Tutorial Notes.
2. Platt J., 1992, “A Generalization of Dynamic Constraints”, Graphical Models and Image Processing”, Vol. 54, No. 6, pp. 516-525.
3. Promayon, E., Baconnier, P., Puech, C., 1996, “Physically-Based Deformations Constrained in Displacements and Volume”, EUROGRAPHICS’96, vol. 15, No. 3, pp. 155-164.

2) *Implementation of constraints to FEM:*

Here, we include a pseudocode that describes how to implement simple boundary conditions to your FEM. Interested readers may find the details of these techniques in suggested references.

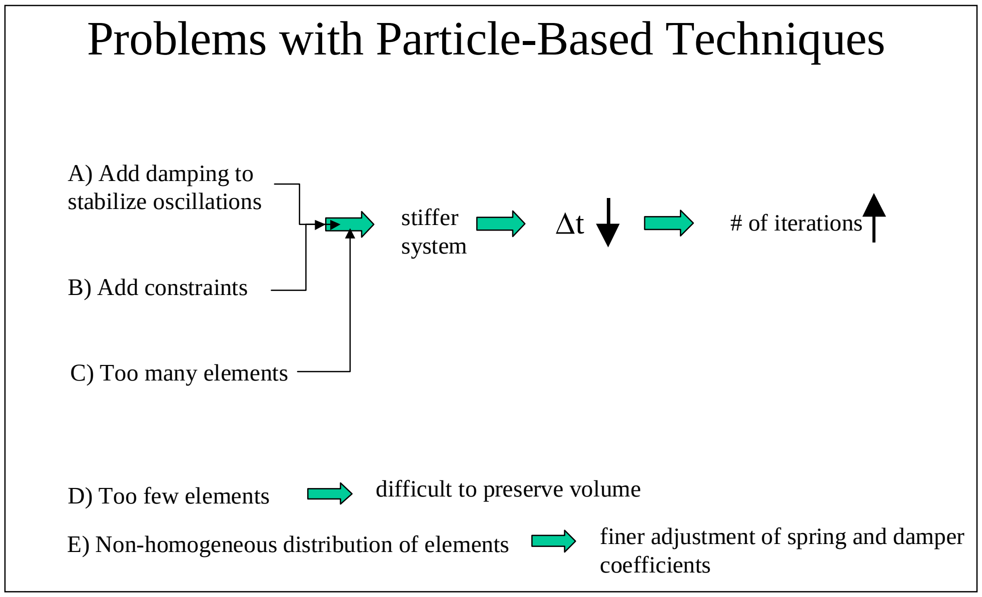
F = K.U (original equation)

FF = KK .U (create a copy of the original system)

```
for j = 1: nn_constrained                              % loop through constraints
        id = bcdof(j);                                 % get the degree of freedom for constraint
        val = bcval(j);                                % get the constrained value
        for i = 1: nn                                  % loop through equations of system
                FF(i) = FF(i) – val*KK(i,id);
                KK(id,i) = 0;
                KK(i,id) = 0;
        end
        KK(id,id) = 1;
        FF(id) = val;
end
```

Once you obtained the modified matrices (KK and FF), solve the modified system for the unknown nodal displacements ($U = KK^{-1}FF$). Then, insert the computed nodal displacements into the original equation (F = KU) to find the applied forces at the nodes. Note that the computations can be simplified if the interactions are point-based only.

Suggested Readings:
1. Huebner, K, Thornton, E., Byrom, T., 1995, “The finite element method for engineers”, John Wiley & Sons, Inc.
2. Kwon, Y.W., Bang, H., 1997, “The finite element method using Matlab”, CRC Press.



Problems with Particle-Based Techniques

A) Add damping to stabilize oscillations

B) Add constraints

C) Too many elements

$\Rightarrow$ stiffer system $\Rightarrow$ $\Delta t \downarrow$ $\Rightarrow$ # of iterations $\uparrow$

D) Too few elements $\Rightarrow$ difficult to preserve volume

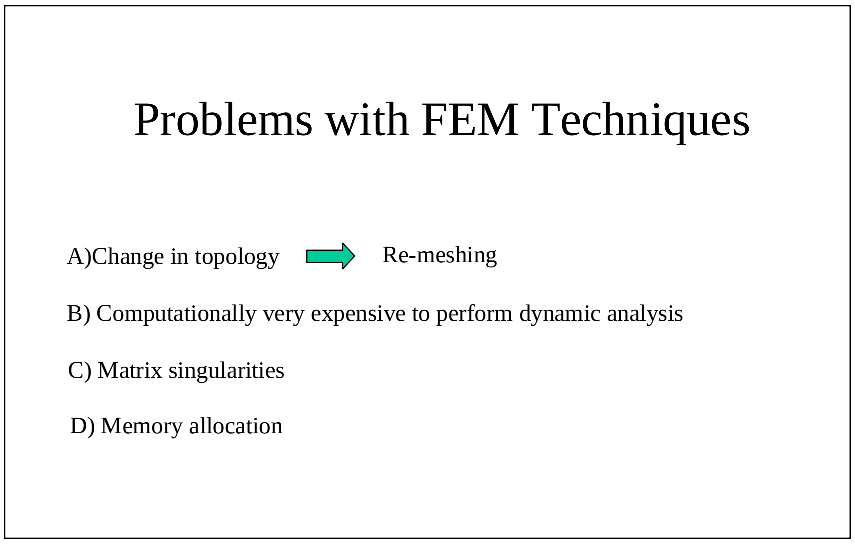
E) Non-homogeneous distribution of elements $\Rightarrow$ finer adjustment of spring and damper coefficients

Some of the problems associated with particle-based systems are listed below

- A damping term needs to be considered to bring the system into a global equilibrium. Increasing damping makes the system *stiffer*. This is going to force us to take shorter time steps to achieve stability.
- Adding multiple constraints leads to a stiffer system. We may need to reduce the "elasticity" of the system to control the deformations. This is, again, going to force us to take shorter time steps to achieve stability.
- Uneven distribution of vertices (nodes) of the 3D model may easily generate unstable interaction forces and non-smooth graphical deformations.
- Note that explicit integration schemes are conditionally stable (see the work done by Barraf and Witkin on implicit techniques, "Large steps in Cloth animation", SIGGRAPH'98)

The following solutions can be proposed for these problems:

- Taking variable time step to improve the stability
- Considering local deformations to reduce the stability problems
- Remeshing or automatic refinement to reduce the stability problems and to make the deformations appear smoother
- Controlling deformation and/or force update rate $(\beta = \frac{\Delta l}{l_0} or \frac{\Delta F}{F_0})$. For example, if ($\beta < \beta_{critical}$) then set $\beta = \beta_{critical}$



Problems with FEM Techniques

A)Change in topology $\Rightarrow$ Re-meshing

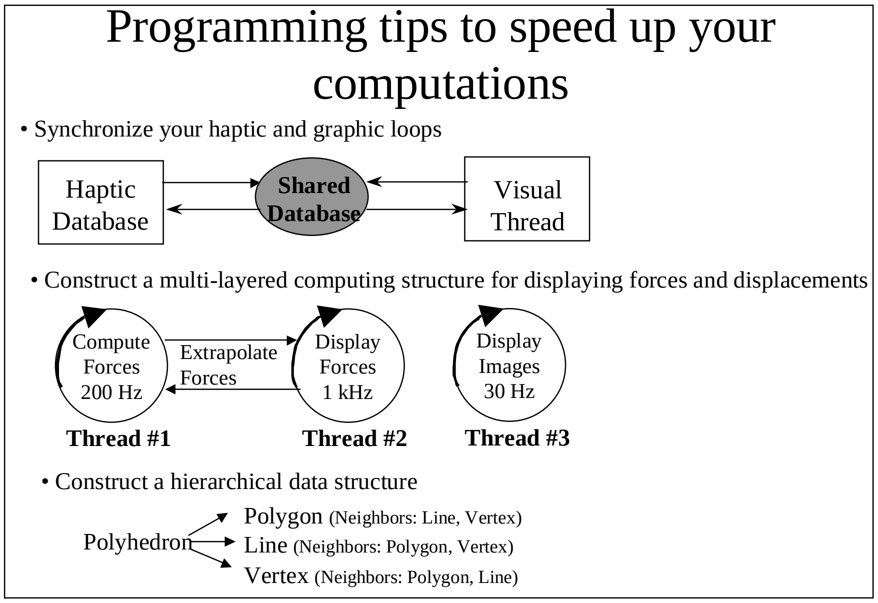
B) Computationally very expensive to perform dynamic analysis

C) Matrix singularities

D) Memory allocation

$$F = KU \qquad U = K^{-1}F$$

In general, finite element models are comphrensive and well suited for accurate computation of deformations. However, it is difficult achive a real-time performance using FEM. Moreover, the addition of haptic feedback increases the complexity of the problem. To achive real-time rendering rates, K^{-1} can be pre-computed and static condensation (i.e. eliminating unwanted degrees of freedom) technique can be implemented. However, the precomputation of K^{-1} is a problem if the topology of object permenantly changes during the interaction. For example, if an object is sliced or cut, it has to be remeshed and the stiffness matrix has to be updated. Moreover, taking the inverse of the K matrix is not trouble free and singularities may occur. Finally, the entries of the matrices need to be allocated wisely to save from the memory.



Programming tips to speed up your computations

• Synchronize your haptic and graphic loops

• Construct a multi-layered computing structure for displaying forces and displacements

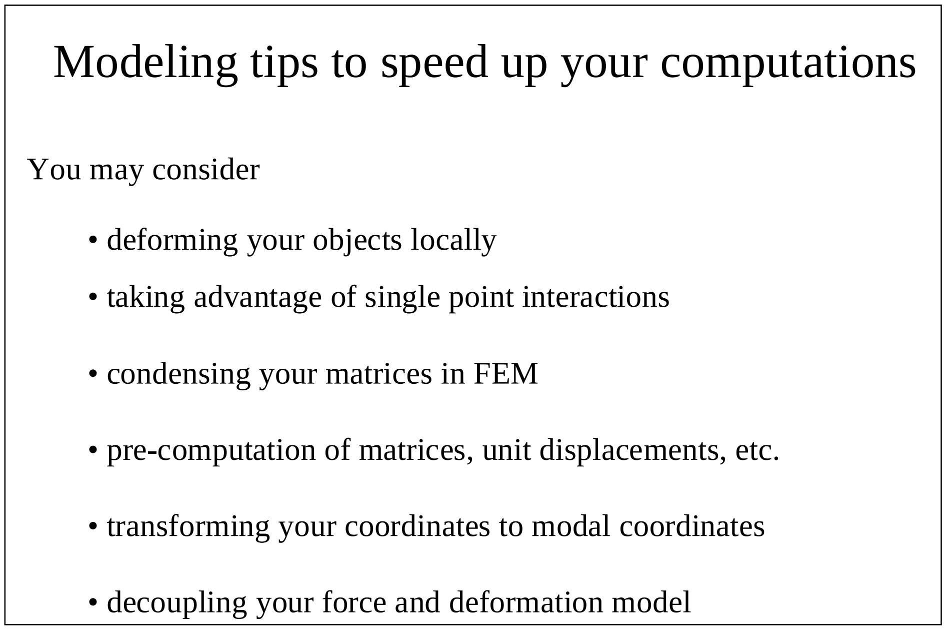
• Construct a hierarchical data structure

Polyhedron
Polygon (Neighbors: Line, Vertex)
Line (Neighbors: Polygon, Vertex)
Vertex (Neighbors: Polygon, Line)

• *Synchronize your haptic and graphic loops:* Software integration of visual and haptic modalities was achieved in an efficient manner by creating a hierarchical database for geometrical properties of objects and by programming with multi-threading techniques. In our simulations, visual and haptic servo loops were separated to achieve faster rendering rates. When displaying visual images, it is known that the update rate should be around 30 Hz to appear continuous. On the other hand, to create a satisfying haptic display, the update rates for sending the force commands to the haptic interface needs to be about 1000 Hz. In order to create a VE that satisfies both requirements and optimally use the CPU power of a computer, the visual and the haptic servo loops need to be separated. That is, we run two loops at the same time, with the graphic loop updated at 30 Hz and the haptic loop updated at 1000 Hz. Since there are two loops running at the same time, there is always a chance a conflict occurs in accessing the shared memory. For example, in the case of simulating deformable objects, changes in geometry require frequent updates of visual and haptic databases in real-time. This will cause a problem if one loop is writing data to the memory and the other loop is reading from there. In order to avoid this situation, we need to synchronize the two loops. The easiest way to synchronize two loops is to create a Boolean flag. When one loop wants to access the shared data, it should check the flag first to see if the data is being accessed by the other loop. If the flag indicates that the shared memory is not being used, the loop can access the data and the flag is set to indicate that the shared memory is currently being used. If the flag indicates that the data is being used by the other loop, the loop waits until the other loop is done. When one loop finishes its operations with the shared memory, it sets the flag back to normal to let other loop access the data.

• *Construct a multi-layered computational architecture:* Although separating haptic and graphic loops, using a client-server model as described in the previous paragraph, is helpful in improving update rates, it may not be sufficient in certain situations. In a typical client-server model for haptic rendering of 3D objects, haptic thread is usually designated as the client and the model computations are performed in this thread. However, physically-based modeling techniques for displaying forces and deformations are computationally expensive and the haptic update rate may drop below the requirement. For example, a real-time dynamic analysis of force-reflecting deformable objects using finite-element techniques is quite difficult with the available computational power. To overcome this difficulty, we suggest a layer between the “computation” and the “display” modules. In this layer, forces can be extrapolated based on the previous force values and their rate of change. Based on this approach, forces can be computed at 200 Hz using a finite element technique, extrapolated in between the computation cycles, and displayed to the user at 1 kHz.

•*Construct a data structure for primitive hierarchy:* We use polygonal models in our simulations. We separate each polyhedron into three types of primitives: vertices, lines (i.e. edges), and polygons. In our database, each primitive has a pointer to its neighboring primitives. The primitive hierarchy helps us to quickly access the neighbors of the primitive when it is necessary. For example, when a simulated tool contacts a primitive of an object in the current loop to modify its coordinates, we know that, in the next loop, the model can only affect the primitives that are in the close neighborhood of the contacted primitive. Neighborhood hirerchy is helpful in simulating force-reflecting deformable objects. For example, forces due to inertial effects can be transferred to all nodes by propagating radially through the neighboring primitives from the contact point.



Modeling tips to speed up your computations

You may consider

- deforming your objects locally
- taking advantage of single point interactions
- condensing your matrices in FEM
- pre-computation of matrices, unit displacements, etc.
- transforming your coordinates to modal coordinates
- decoupling your force and deformation model

Number of computations is significantly important in simulating force-reflecting deformable objects in virtual environments. Most of the time, the developer needs to reduce the # of computations or to make simplifications in the model in order to achieve real-time rendering rates. Here, we suggest a few tips in this regard:

1) *deforming your objects locally*

```
r = |vertex[i].coord - Collision Point |;
if ( r < $R_{deformation}$ )
    vertex[i].frozen = yes;
```

2) *taking advantage of single point interactions*

a) "if the force is applied to a single node" in FEM

$$U = K_i^{-1} F_i$$

where, "i" is the i-th column of K^{-1} matrix and i-th entry of force vector.

b) "if the force is applied to a single node" in FFD model (refer to spline-based modeling section)

For single point manipulation, the solution reduces to the following simple form

$$\Delta P = \frac{B^T}{\sum_i (b_i)^2} \Delta Q$$

where, b_i's are the elements of the B matrix.

3) *condensing your matrices in FEM*

a) e.g.: construct your matrix using volume elements, but solve the equations for surface elements

b) e.g.: eliminate unwanted degrees of freedom such as rotational dof.

$$\begin{bmatrix} K_{MM} & K_{MS} \\ K_{SM} & K_{SS} \end{bmatrix} \begin{bmatrix} U_M \\ U_S \end{bmatrix} = \begin{bmatrix} F_M \\ F_S \end{bmatrix}$$

$$\left[\overline{K}\right]\left[U_M\right] = \left[\overline{F}_M\right]$$
$$\left[\overline{K}\right] = \{K_{MM} - K_{MS} K_{MM}^{-1} K_{SM}\}$$
$$\left[\overline{F}_M\right] = \{F_M - K_{MS} K_{SS}^{-1} F_S\}$$

where, the subscript M denotes the masters and the subscript S denotes the slaves which are to be eliminated.

4) *pre-computation in FEM*

a) compute K^{-1} in advance
b) compute displacements for a given unit force at each node and then apply superposition

5) *transforming your coordinates to modal coordinates (dynamic analysis)*

If you would like to simulate the inertial properties of force-reflecting deformable objects, it may be worthwhile to consider a modal analysis. In modal analysis, you transfer your coordinates to modal coordinates to decouple your differential equations. This will enable you to obtain the explicit form of the governing equation for each node. Moreover, you can also reduce the dimension of the system by picking the most significant modes and re-arranging your mass, damping, and stiffness matrices (i.e. modal reduction).

a) transforming to modal coordinates (coupled diff. Eqs. -> uncoupled diff. Eqs.)

$$M\ddot{U} + B\dot{U} + KU = F$$
$$U(t) = \Phi X(t)$$
$$\Phi^T M\Phi\ddot{X} + \Phi^T B\Phi\dot{X} + \Phi^T K\Phi X = \Phi^T F$$
$$\ddot{X} + \Phi^T B\Phi\dot{X} + \Omega^2 X = \Phi^T F$$

where,

$\Phi = [\varphi_1, \varphi_2, \varphi_3, ..., \varphi_n]$ $\quad$ $\Omega^2 = diag(\omega_i^2)$

Φ is a modal matrix whose columns are eigenvectors of ($M^{-1}K$) and Ω^2 is a diagonal matrix which stores the eigenvalues on its diagonals. Observe that $\Phi^T B\Phi$ is not a diagonal matrix (i.e. we still have a system of uncoupled differential equations). Assume that damping matrix is propotional to mass and stiffness matrices ($B = \alpha M + \beta K$, where α and β are constants. Then the equations take the following form:

$$\ddot{X} + \Delta\dot{X} + \Omega^2 X = \Phi^T F \qquad \text{(a set of uncoupled diff. eqs.)}$$

where, $\Delta = \text{diag}(2\omega_\text{i}\zeta_i)$ and ζ is modal damping factor.

b) modal reduction (eliminate high frequency modes)

This technique involves the selection of dominant modes and elimination of high frequency modes. To achieve this, the eigenvalues of the system are listed in incerasing order, and then the columns of the ϕ matrix are rearranged according to this order to construct a reduced order system. Note that the first six nodes of the eigen-matrix represent the rigid body modes.

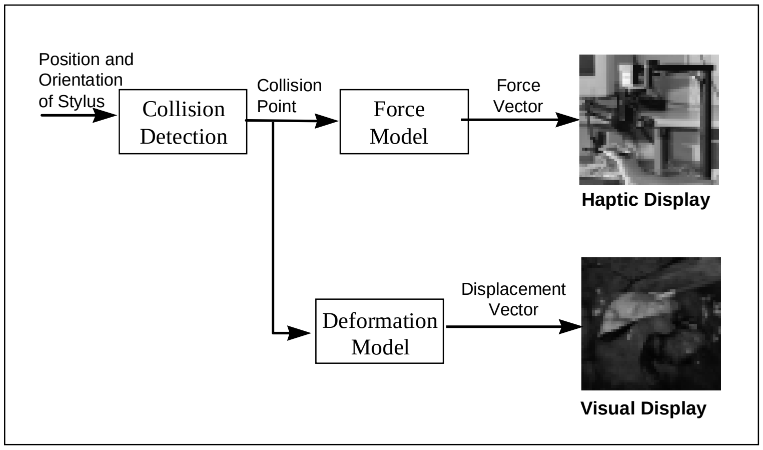
$$\Phi = [\varphi_1, \varphi_2, \varphi_3, \varphi_4, \varphi_5, \varphi_6, ..., \varphi_n]$$

<u>Suggested Readings:</u>

1. Bathe, K., 1996, “Finite Element Procedures”, Prentice Hall, New Jersey.
2. Shabana, A., 1996, “Theory of Vibration”, Springer-Verlag.
3. Pentland, A., Williams, J., 1989, “Good Vibrations: Modal Dyanmics for Graphics and Animation”, SIGGRAPH Proceedings, Vol. 23, No. 3, pp.215-222.

6) *loose coupling of force and displacement models:*

One can loosely couple the deformation model with the force model to simulate the nonlinear material characteristics of deformable objects. To implement this idea, the information returned by the collision-detection module (collision point and depth of penetration) can be independently used by “deformation” and “force” models. This technique, for example, can be used to simulate the “nonlinear force” characteristics of soft tissues. Since the developed tissue models for simulation purposes are usually linear, nonlinear force-displacement characteristics of organs would not be simulated using these models. However, a nonlinear force profile constructed using the experimental measurements can be used to reflect nonlinear forces to the user while a smooth deformation profile is displayed graphically using the FFD technique.



The concept of loosely coupling force and displacement models. To implement this idea, we independently use the collision point and depth of penetration in deformation and force models following the detection of collision.