A Layered Approach to Deformable Modeling and Animation

Clint Chua and Ulrich Neumann
{chua | uneumann}@usc.edu
Computer Science Department
Integrated Media Systems Center
University of Southern California

Abstract

Our approach integrates three mechanisms needed to model and animate deformable objects: controlling mechanisms, geometric surface deformation, and mesh refinement. Most approaches focus either modeling physically correct behavior or alternative representations for deformable models. This results in a set of method specific algorithms that best represents a particular class of deformable objects. By encapsulating each process, our system introduces the interface that allows us to integrate existing controllers in a modular fashion. We demonstrate this in our system by instantiating hardware accelerated free form deformation for geometric deformation and midpoint subdivision for mesh refinement. Finally, we discuss the available options for controlling mechanisms and show how this approach leads to a generalizable framework.

1 Introduction

Modeling and animating a deformable object often involves the simultaneous use of physical simulators, surface deformation and mesh refinement. Thus, to incorporate a deformable object into a virtual environment, these three processes must be integrated into a system. Since existing approaches to deformable objects are either material specific [32] or method specific [13], this limits the class of deformable objects in any one system. Implementing several deformable modeling approaches is difficult since there is no common interface among the different approaches. This paper shows how three processes can be integrated in one deformation system that supports a wide range of options for each component.

The first process is the controlling mechanism. This refers to the process that controls the deformation of the object. This often is a physical simulator but in some cases can be a user interface to the surface deformation engine. The latter case is common when artists require full control over the object rather than having the object respond in a physically correct manner.

The second process is the surface deformation engine. This process alters the surface geometry of the deformable object. In some approaches, the physical simulator and the surface deformation engine cannot be differentiated. Most physical simulators directly alter the surface geometry. Combining the controller with the surface deformation process results in method specific deformable systems. We separate these processes in order to enforce a standard structure in our system that allows multiple controllers to be used with the same deformation engine.

The third process is the mesh refinement system. Although models could be represented using several representations, triangular meshes are common and efficient for rendering. Therefore, the accuracy of the model representing the deformed object can depend on the amount of refinement that is applied to the model. This process ranges from re-meshing, commonly used with implicit surface representations, to subdivision surfaces for triangular or quadrilateral meshes. Some deformation techniques ignore this process by using a fixed but highly tessellated mesh. This is inefficient if some portions of the mesh remain planar while other portions have high surface curvature.

Our system implementation focuses on the second and third processes since we intend to use existing controlling techniques. A natural extension to our approach is its use as a framework for general deformable modeling and animation with a variety of controllers.

The next section discusses related work in deformable modeling. Sections 3 through 5 detail our system concept. Section 6 discusses available controlling technologies. Section 7 describes our implementation, and the last section summarizes the utility and features of our system.

2 Related Work

There is a lot of research done in physically correct deformable modeling. Physical simulation, ranging from mass-spring models to finite element methods [13], is often

used in deformable modeling as a controlling mechanism. Gibson and Mirtich give a comprehensive survey of this in [3]. In contrast to prior work, space-time adaptive sampling applied to finite element methods achieves physically correct real-time deformations [1, 2].

Purely geometric surface deformation engines have not been given much attention. The idea of a geometric deformation primitive was first introduced in [4] and later developed into free form deformation (FFD) [5], extended FFD (EFFD) [6] and FFD with lattices of Arbitrary Topology [7]. WIRES [8] introduced a new geometric deformation primitive. Instead of deformation via a bounded region of space that characterizes most geometric deformation primitives, WIRES utilized axial deformation, deformation of space around a curve. WIRES is also able to mimic the functionality of FFD and some of its variants.

Implicit surfaces are alternate representations of deformable objects [9,10,11]. Another alternative representation called Smoothed Particles is introduced in [12]. Although these alternatives can represent deformable objects, the rendering pipeline is optimized for triangular meshes. The deficiency of accurate modeling using meshes is compensated by the introduction of mesh refinement primitives such as Catmull-Clark surfaces [15] and other surface interpolatory schemes [16, 17, 18, 19, 27]. The idea behind all of these methods is inserting new triangles or quadrilaterals based on a weighting of a local cluster of vertices. These techniques achieve smooth and continuous surfaces when recursively applied to the mesh.

New techniques [23, 24] deal with two processes: controlling mechanisms and mesh refinement. These systems validate the trend of deformation systems converging toward the three processes identified in this paper. While their systems are an improvement, these still focus primarily on method specific deformation systems.

3 System Overview

Our approach divides the deformation system into three layers, as shown in Figure 1. The first layer, called the controlling mechanism, consists of either purely user-based control or physically based control. The responsibility of this layer is to direct the surface deformation layers either through physical simulation or direct user interaction. Since varied control mechanisms are feasible, and our choice is not limited, our discussion does not focus on this layer.

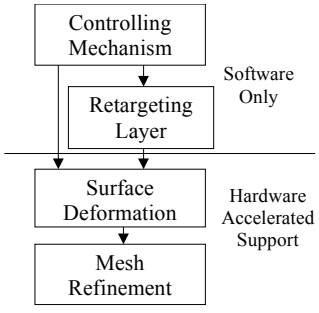

Figure 1 System Overview

The second layer is the surface deformation system. This layer represents a geometric deformation method. Our implementation uses the Extended Free-Form Deformation (EFFD) primitive since FFD is popular, computationally inexpensive, and easy to implement. Among all the variants of the FFD, EFFD retains the mathematical simplicity of the FFD and yet is able to use more complex lattice shapes compared to the original FFD method. In addition, we show in our previous work that the EFFD technique can be hardware accelerated in front of a standard OpenGL pipeline [28]. A description of EFFD is outlined in Section 4.

The third layer is the mesh refinement system. We chose to work with a triangular mesh representation since graphics pipelines render triangular meshes directly. In order to maintain representation accuracy while minimizing the total number of triangles in the mesh, we employ an adaptive midpoint subdivision operator for mesh refinement. For all deformed states, this allows us to process a minimal number of triangles. We chose midpoint subdivision since it is simple and can be hardware accelerated as proposed by Bischoff and Kobbelt[20]. The basics of midpoint subdivision are introduced in Section 5.

We add a retargeting layer between the controlling mechanism and the surface deformation layer. In our approach, physical controllers no longer displace vertices of the mesh directly. Thus, a retargeting layer is needed to convert the output of the physical controller and map it to an appropriate input to the EFFD. The details of the retargeting layer are discussed further in Section 6.

4 Surface Deformation

4.1 Background

The analogy behind all types of FFDs is embedding an object into a piece of gelatin, and as the gelatin deforms, the embedded object deforms with it. The different FFDs only vary the initial shape of the gelatin in which the object is embedded. A standard FFD works solely with rectangular parallelepipeds. Arbitrary topology FFDs allow any control lattice. The EFFD, while not as versatile as the arbitrary topology FFD, can work with cylinders, spheres and other regular shapes more complex than a rectangular parallelepiped, while still using the efficient deformation equations of the standard FFD.

EFFD operation is divided into 3 steps. The first step embeds the object in the initial control lattice and computes the parameterized coordinates of the object. The next step moves the control lattice points to new locations, thus deforming the enclosed region of space. The last step calculates the deformed positions of every object point based on the new locations of the control points. The deformed object is then ready for rendering.

The embedding process starts with a lattice bounding a volume of space to be deformed. An object is embedded within the lattice by computing a transformation of coordinate systems from the object coordinate system to the

local lattice coordinate system. The embedding process requires a solution to a system of non-linear equations. This system of non-linear equations is defined by the equations of deformation so we will first introduce these equations and then show how the embedding is done.

Assuming that each object point $X=(x,y,z)$ has a parameterized local coordinate (s,t,u) and $0 \le s,t,u \le 1$, then with the set of control points P, we calculate the deformed position q with

$$q_{i,j,k}(s,t,u) = \sum_{l,m,n=0}^{3} P_{i+l,j+m,k+n} B_l(s) B_m(t) B_n(u) \qquad (1)$$

where $P_{i,j,k}$ is the *ith, jth, kth* control point and the *Bs* are the uniform cubic B-spline blending functions. Thus, given a set of parameterized local coordinates, we evaluate the above equation to get the set of deformed points based on the new positions of the control points.

For the embedding process, we need to derive the system of non-linear equations to determine the parameterized local coordinates. From equation (1), we can derive the system of non-linear equations for the (s,t,u) parameterization of an object point X. The intuition behind the embedding procedure for the EFFD is that it is the inverse of deformation of the object in the initial control lattice to a rectangular parallelepiped control lattice. Hence, the initial EFFD control lattice shape is constrained by the existence of a mapping or morph between it and a standard rectangular parallelepiped. The morph must not incur any space folding, that is, it must be invertible. From this we can see why the EFFD is not appropriate for lattices of arbitrary topology since the morph from the arbitrary lattice to the rectangular parallelepiped may not exist. To derive the system of non-linear equations, we set the object point X equal to q, the deformed point in Equation (1). The goal then is then to find a parameterization (s,t,u) that satisfies this equation. In order to find (s,t,u), we rearrange Equation (1) to obtain

$$\sum_{l,m,n=0}^{3} P_{i+l,j+m,k+n} B_l(s) B_m(t) B_n(u) - X_{i,j,k} = 0 \qquad (2)$$

where Xi,j,k is the object point within the deformable region of $P_{i,j,k}$ to $P_{i+3,j+3,k+3}$. Since this is a 3-D vector equation, we have one equation from each dimension, and 3 unknowns, namely (s,t,u). We use a Newton-Raphson root-finding method with initial guess 0.5 to find (s,t,u). For a more detailed explanation of the EFFD process and properties, refer to [5, 6, 7, 28].

4.2 Lattice Translation Table

Our formulation of the FFD equation (Equation 1) uses the uniform cubic B-spline basis functions. The parameterized point depends on a 4x4x4 set of local lattice points that surround the object point. A 4x4x4 lattice grid is required to deform a cell volume within the lattice, as shown in Figure 2. For more complicated lattices with more than one cell volume, the process of embedding an object point also entails locating which cell volume

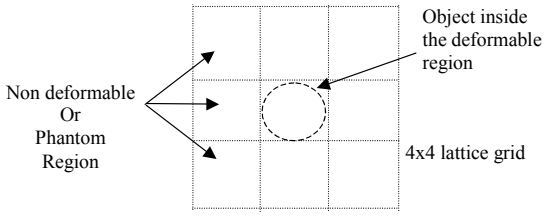

Figure 2: A 2-D view of the lattice grid showing the 4x4 set of control points for a single cell of deformation, the region of deformation in the center of the bounded region, and the non-deformable regions surrounding it.

controls the space that the object point is located.

Since the lattice shape is not always a rectangular parallelepiped, we propose an indexing structure for fast object-point to cell-location translation called the Lattice Translation Table (LTT) shown in Figure 3. The LTT is an axis aligned bounding box of the lattice that is divided into regularly spaced cubes. For each cube that intersects with the deformable area of the lattice, indices to the deformable area are kept in each cell. If more than one deformable region occupy a cube, only one index is kept since the appropriate cube can be deduced as an extrapolation from the results of the embedding process. This extrapolation mechanism will be discussed in the next section.

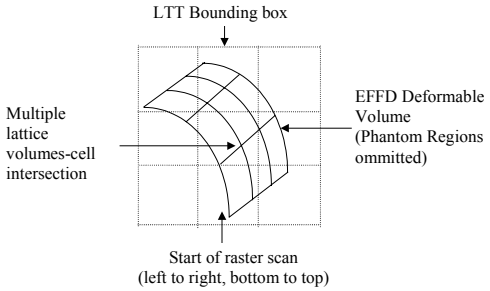

Figure 3: The Lattice Translation Table

The creation of the LTT is done in a raster-order sweep of each cube in the LTT. A set of test points in each cube is chosen, typically the center and the corner points of the cube, then these points are tested for their positions within the lattice volume by parameterizing them in the lattice. If the parameterization yields coordinates such that $0 \le s,t,u \le 1$ then we found a lattice volume-cube intersection. A good starting point for the raster sweep is to use the location of the control point at $(0,0,0)$ in the lattice grid and then use the cube that contains the control point. Notice that some cubes that contain lattice volume-cube intersection may be missed using this algorithm if the lattice volume is concave (e.g. an extruded star shape). One solution is to add more test points in each cube. Another option is to grow the existing cubes to encompass the entire volume of the lattice. This latter method takes the existing set of lattice volume-cube intersections and tests the neighboring cubes that do not have lattice volume-cube intersections.

Neighboring cubes are chosen for testing based on control points that have common edges with other control points that lie inside cubes with lattice volume-cube intersections.

The LTT is created as a pre-processing step and each LTT is associated with a specific lattice. This means that a whole library of lattices with its associated LTT files can be created beforehand without prior knowledge of the object to be deformed. Once the lattice and LTT structures are created, embedding the object points involves an LTT lookup of the lattice index belonging to the LTT cube that the object point is in. These indices are then used in the Newton-Raphson root finding method for each point.

4.3 Embedding and Extrapolation

The embedding process described in section 4.1 can also be used to extrapolate the correct cell that an object point belongs to if a good initial guess was provided. Since overlapping deformable volumes exist in an LTT cube, only one lattice index in the LTT can be used as a starting guess for all points within an LTT cube.

Since the parameterized coordinate is constrained such that $0 \leq s,t,u \leq 1$, coordinates that do not satisfy this constraint mean that the object point does not belong to the lattice cell and also that the object point lies in the direction of the axis that does not satisfy the constraint. In other words, if $s < 0$ or $s > 1$ then the lattice cell that is in the $x-1$ or $x+1$ position relative to the current cell will contain the object point (Figure 4).

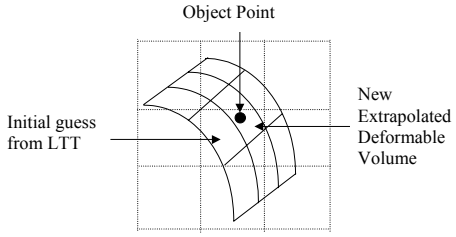

Figure 4: Extrapolation procedure

Thus the process of embedding and extrapolation is as follows. First, the initial guess, (x,y,z), of the lattice cell index is taken from the LTT. The object point is parameterized against the lattice cell. If it satisfies the constraint such that $0 \leq s,t,u \leq 1$, then the object lies in that lattice cell with parameterized coordinates, (s,t,u). Otherwise, the lattice cell index (x, y, z) is incremented or decremented depending on s, t, u less than zero or greater than one, respectively.

5 Mesh Refinement

5.1 Midpoint Subdivision

Midpoint subdivision is the process of refining a triangular mesh by subdividing existing triangles. Vertices are inserted at the midpoint of each edge of the triangle and a new triangle is created as shown in Figure 5. Midpoint subdivision is often applied uniformly to the entire mesh. This has both advantages and disadvantages. One advantage is that several recursive applications of midpoint subdivision easily create a smooth finely tessellated mesh. The disadvantage of uniformly applying midpoint subdivision is the exponential growth in the number of triangles in areas that do not need the subdivision. The solution is to apply midpoint subdivision only to parts of the mesh that require subdivision based on a curvature or surface error metric.

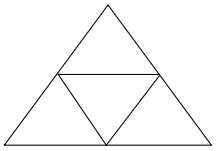

Figure 5: Subdivided Triangle

Adaptive subdivision gives rise to a problem with boundary triangles. Options for dealing with boundary triangles are shown in Figure 6.

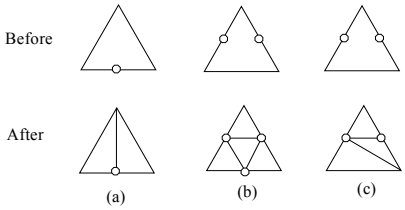

Figure 6:
(a) Singe point case.
(b) Re-subdivide
(c) early termination

Two cases arise when performing adaptive midpoint subdivision. A boundary triangle will either have one or two new vertices added to its edges. In the single point case shown in Figure 6a, the triangle is cut in half. In the two-point case shown in Figure 6b and 6c, there are two ways to subdivide the triangle. The first is to add the missing third point and apply midpoint subdivision (Figure 6b). The other is to split it into three triangles as shown in Figure 6c.

The first method has the advantage of maintaining the uniformity of subdivision, however, in introducing the third vertex, the triangle is changed from being a border triangle to a midpoint-subdivided triangle. This method not only expands the boundary triangle region but also causes a cascading midpoint subdivision effect whenever the added vertex also lies on another boundary triangle as shown in Figure 7.

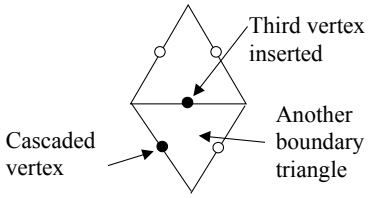

Figure 7: Using the Re-subdivide technique adds the third vertex on the top triangle. This adds the second vertex for the lower triangle and causes another application of the Re-subdivide technique and thus the cascaded vertex.

The second method does not produce this cascading effect since no new vertices are added. It has the drawback that long thin triangles may be produced but degenerate configurations of boundary triangles can cause cascading, leading to the introduction of unnecessary triangles.

5.2 Representation

We used a forest of trees representation for the subdivided triangles as shown in Figure 8. Instead of re-evaluating the FFD equation (Equation 1) from the parameterized coordinates and lattice indices, object coordinates are kept if the lattice volume was not deformed. Pointers to neighboring triangles are kept in order to perform boundary triangle subdivision. If a triangle is beside a midpoint subdivided triangle, then the neighboring triangle is either subdivided into two or three triangles, depending on the number of new vertices introduced by other midpint subdivided triangles. Pointers to subdivided triangles are used to keep track of the current level of subdivision. In addition, reversing the subdivision process is possible since we still keep the original triangle in this representation.

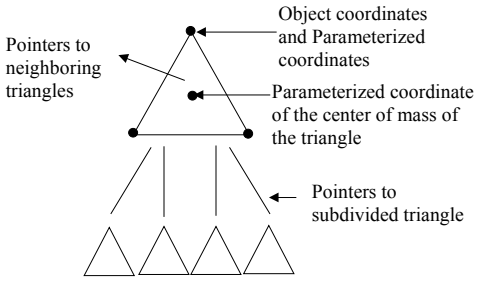

Figure 8: Tree representation

The root of each tree corresponds to a triangle in the original undeformed mesh. The depth of the tree depends on the number of applications of subdivision a particular triangle has undergone. Traversing each tree and rendering its leaf nodes renders the model.

5.3 Subdivision Criteria

As the mesh is deformed, the accuracy of the deformed model is generally diminished. A fidelity criterion determines where additional triangles are needed. We use the distance between the undeformed and the deformed center of mass of the triangle as a metric that is simple and fast to evaluate.

During the pre-processing stage of parameterizing the object coordinates, the center of mass of each triangle's vertices is also parameterized. Whenever a triangle undergoes deformation, the center of mass in world coordinates is calculated based on the current configuration of the triangle. The original center of mass is recalculated using the parameterized coordinates and Equation 1. If the distance between these two coordinates exceeds a threshold, the triangle is subdivided. (To avoid a square root calculation, use the square of the distance.)

6 Controlling Mechanisms

In order to deform the object, we need to manipulate the control points. Control points generally number significantly fewer than the deformed mesh vertices, however control of these points is still nontrivial. This section reviews some geometric and physically based controllers best suited for controlling FFD lattices.

6.1 Radial Basis Function

A Radial Basis Function (RBF) is a popular multidimensional interpolation method. By controlling a small subset of FFD control points, the RBF method can interpolate new positions of the remaining control points. For example, the corner control points of the lattice are chosen as the interpolation points of the RBF from which the internal control point locations are derived.

The RBF is a system of linear equations of the following form.

$$y_j = \sum_{i=1}^{N} c_i \left\| x_j - x_i \right\| \quad j = 1,\ldots,N \tag{3}$$

where y_j is a vector containing a single dimension of the corner control point location, ci is an unknown weighting factor, $||x_j-x_i||$ is the pair-wise Euclidean distances between each of the corner control points, and N is the total number of corner control points.

Interpolating is done as follows. Define a symmetric matrix H such that $H_{ij} = ||x_j-x_i||$, the pair-wise Euclidean distances of the corner control points. Recover the weighting factors c by solving the equation $c=H^{-1}y$. The new location of the internal control points is calculated by substituting the internal control point location for x_j in Equation 3. This whole process is done for all three dimensions, but since H^{-1} is the same for each dimension, the inverse is only calculated once. For more the details on RBFs, refer to [29].

6.2 Skeletal Animation Systems

Skeletal animation systems embed a movable skeleton inside an object. Skeleton motions, influence the surrounding mesh. As the name implies, skeletal animation systems are a form of axial deformation. The skeleton bones form the axis and vertices move to preserve their distance from an axis. Two or more bones may influence vertices near joints.

Skeletal animation can control axial deformations of an FFD lattice. Analogous to a skin that wraps around a bone, the bone can control an FFD lattice and multiple bones at joints can control a single control point. In turn, the object embedded in the FFD lattice moves according to the

skeletal deformation. For more details on skeletal systems, refer to [30].

6.3 Directly Manipulated Free-Form Deformation

While global deformations are easily obtained by direct manipulation of FFD control points, attaining specific detailed deformations can require unintuitive configurations of control points. Animators and modelers often prefer to manipulate the surface directly. This requires the system to hide the indirect and automatic manipulation of the control points from the user. This is called Directly Manipulated Free-Form Deformation (DMFFD). The user manipulates a set of vertices on the surface. moving them to their desired final locations. The DMFFD system calculates the best configuration of control points to produce the desired vertex positions. Determining the best control point configuration can be posed as either an underdetermined or overdetermined system of non-linear equations, depending on the number of vertices and the total number of control points. Standard solution methods employ either naïve pseudo-inverses or Householder's QR factorization. For details on solving these equations, refer to [25, 26].

A more important aspect of DMFFD is how it can be used as a retargeting layer as mentioned earlier in Section 3. Since most physically based controllers manipulate the mesh directly, DMFFD is the best method by which the output of a physically based controller can be retargeted to controlling the FFD control points. This means that no changes are needed in the physically based controller algorithm because of DMFFD. In addition, since DMFFD handles general direct surface manipulation, it is conceivable to design a whole new set of controllers on top of DMFFD, be it physically based or non-physically based.

6.4 Mass Spring Systems

Mass-spring systems are often used in animation systems. Their applications range from facial animation to simple deformable objects. In addition, they are easy to implement and computationally inexpensive for lattices of up to a few hundred points. As external forces, such as gravity, are applied to one or more of the mass elements, a simulation of the mass and spring interactions is computed until an equilibrium state is reached.

The application of a mass-spring system as an FFD controller is used in [22]. Each control point is considered a mass element while the lattice topology represents the spring network that ties the masses together. With this arrangement, a user can either apply forces to the FFD lattice or manipulate the mass elements to control the FFD control points. This system produces physically plausible dynamics and responses for the object.

6.5 Finite Element Methods and Other Physical Controllers

The finite element method (FEM) can be thought of as a generalized extension to the mass-spring model. An object is represented by a set of discrete finite elements. The interconnections between finite elements define how these elements interact. Instead of using Hooke's spring laws, FEMs use equations that model the level of potential energy within the object and the physical simulation aims at minimizing the total potential energy. FEMs give the most physically accurate representation of elastic deformation but they tend to be computationally expensive. Recently, techniques achieving real-time simulations using FEMs include [1, 2, 14, 23, 24].

Since FEM control mechanisms produce the best representations of elastic objects, they are desirable to incorporate as controllers. As discussed in Section 6.3, DMFFD can serve as a retargeting layer to translate the output of FEMs into FFD control point configurations. The physical simulator and representations used by FEMs remain the same, only the final output object configuration is re-targeted. This approach allows our deformation and subdivision method to utilize a rich library of physical controllers.

Another method of integrating physical controllers into FFD systems was proposed in Dynamic Free Form Deformations [31]. This method defines a parametric space of different lattice configurations representing basic operations performed on the object such as bending and shearing. Lagrangian dynamics is used to simulate both locomotion and internal strain energy. Deformation is calculated as a linear combination of the several operation parameters based on the calculated strain energy of the object.

7 Results

We implemented Sections 4 and 5 in software running on a Pentium II 450Mhz PC. An RBF controller is used as the controlling mechanism for our experiments. Four models ranging in complexity and surface properties serve as test cases. We tested bending along the y-axis of the object and measured the amount of subdivision that occurred. The results are summarized in Table1 and are shown in Plate 1.

	Triangle Count	
	Undeformed	Deformed
Cube	12	176
Bookcase	108	1368
Chair	1232	1407
Venus model	1396	1718

Table 1: Triangle Counts

The first two cases show the greatest amount of subdivision since the original mesh is composed of flat

surfaces. The chair and Venus model only added a small number of triangles. Triangles were added mostly to the stem of the chair and the waist area of the Venus model since these are the apex of the bending operation and thus experience the highest curvature change. This also shows the benefit of adaptive subdivision since additional triangles are only added where necessary.

8 Conclusion

We developed a unified deformation system by identifying common processes of deformation systems: controlling mechanisms, surface deformation and mesh refinement. Emphasis was placed on the last two layers by discussing our selection of EFFD and midpoint subdivision as components in our system. Hardware acceleration of these two components is feasible, leading us to believe that they can form the foundation of real-time interactive deformable modeling and animation. We describe how to integrate a rich library of controlling mechanisms, ranging from RBFs to FEMs, into this system.

9 References

[1] G. Debunne, M. Desbrun, M.P. Cani, A. Barr. Dynamic Real-Time Deformation using Space and Time Adaptive Sampling. To appear in SIGGRAPH 2001.

[2] G. Debunne, M. Desbrun, M.P. Cani, A. Barr. Adaptive Simulation of Soft Bodies in Real-Time. Computer Animation, pp. 17-24, 2000

[3] S. Gibson, B. Mirtich. A survey of Deformable Modeling in Computer Graphics. MERL Technical Report, TR-97-19, November 1997.

[4] A. Barr. Global and local Deformations of Solid Primitives. SIGGRAPH July 1984 , pp 21-30.

[5] T. Sederberg and S. Parry. Free-Form Deformation of Solid Geometric Models. SIGGRAPH 1986, pp. 151-160.

[6] S. Coquillart. Extended Free-Form Deformatin: A Sculpting tool for 3D geometric modeling. SIGGRAPH 1990, pp. 187-196.

[7] R. MacCraken and K. Joy. Free-Form Deformation with Lattices of Arbitrary Topology. SIGGRAPH 1996, pp. 181-188.

[8] K. Singh and E. Fiume. Wires: A Geometric Deformation Technique. SIGGRAPH 1998, pp. 405-414.

[9] M. Desbrun and M.P. Gascuel. Animating Soft Substances with Implicit Surfaces. SIGGRAPH 1995, pp. 287-290.

[10] M.P. Gascuel and M. Desbrun. Animation of Deformable models using implicit surfaces IEEE Transactions on Vision and Computer Graphics. March 1997.

[11] M. Desbrun and M.P. Gascuel. Active Implicit Surfaces for Animation. Graphics Interface 1998

[12] M. Desbrun and M.P. Gascuel. Smoothed Particles: A new paradigm for animating highly deformable bodies. 6th Eurographics Workshop on Animation and Simulation 1996.

[13] D.Terzopoulos, J. Platt, A. Barr, K. Fleischer. Elastically Deformable Models. SIGGRAPH 1987, pp. 205-214.

[14] D. James and D. Pai. ArtDefo: Accurate Real Time Deformable Objects. SIGGRAPH 1999, pp. 65-72.

[15] E. Catmull and J. Clark. Recursively generated B-spline surfaces on arbitrary topological meshes. Computer Aided Design 10(6): 350-355, 1978.

[16] N. Dyn, D. Levin, J. Gregory. A Butterfly subdivision scheme for surface interpolation with Tension Control. Transaction on Graphics Vol. 9, No. 2, April 1990, pp. 160-169.

[17] N. Dyn, D. Levin, J. Gregory. A 4-point Iterpolatory subdivision scheme for curve design. Computer Aided Geometric Design 4(1987), pp. 257-268.

[18] L. Kobbelt. Variational Subdivision Schemes. Computer Aided Geometric Design 13(1996), pp 743-761.

[19] D. Zorin, P. Schroder, W. Sweldens. Interpolating Subdivision for Meshes with Arbitrary topology. SIGGRAPH 1997, pp. 259-268.

[20] S. Bischoff, L. Kobbelt, H.P. Seidel. Towards Hardware Implementation of Loop Subdivision. SIGGRAPH-EUROGRAPHICS Workshop On Graphics Hardware 2000, pp. 41-50.

[21] K. Muller, and S. Havemann. Subdivision Surface tesselation on the Fly using a versatile Mesh Data Structure. Eurographics Vol. 19, No. 3, pp. 151-159, 2000.

[22] J. Christensen, J. Marks, J. Ngo. Automatic motion synthesis for 3D mass-spring models. Visual Computer, Vol. 13, pp. 20-28, 1997.

[23] K. McDonnell, H. Qin. Dynamic Sculpting and Animation of Free-form subdivision solids. Computer Animation 2000, pp. 138-145.

[24] C. Mandal, H. Qin, B. Vemuri. Dynamic Modeling of Butterfly subdivision sufaces. Transactions on Visulaization and Computer graphics July-September 2000, pp. 265-287.

[25] J. Gain, Virtual Sculpting: An Investigation of Directly Manipulated Free-Form Deformation in a Virtual Environment. Masters Thesis, Rhodes University. February 1996.

[26] W. Hsu, J. Hughs and H. Kaufman. Direct Manipulation of Free-Form Deformations. SIGGRAPH 1992, pp. 177-184.

[27] T. DeRose, M. Kass and T. Truong. Subdivision Surfaces in Character Animation. SIGGRAPH 1998, pp. 85-94.

[28] C. Chua and U. Neumann. Hardware Accelerated Free Form Deformation. SIGGRAPH-EUROGRAPHICS Workshop On Graphics Hardware 2000, pp. 33-39.

[29] T. Poggio and F. Girosi. A Theory of Networks for Approximation and Learning. A.I. Memo No. 1140. Artificial Intelligence Lab, MIT, Camabridge, MA, July 1989.

[30] J. P. Lewis, M. Cordner, N. Fong. Pose Space Deformation: A Unified Approach to Shape Interpolation and Skeleton-Driven Deformation. SIGGRAPH 2000, pp. 165-172.

[31] P. Faloutsos, M. van de Panne, D. Terzopoulos. Dynamic Free-Form Deformations for Animation Synthesis. IEEE Transactions on Visualization and Computer Graphics. Vol. 3, No. 3, pp. 201-214, July-September 1997

[32] D. Baraff, A. Witkin. Large Steps in Cloth Simulation. SIGGRAPH 98, pp. 43-54.

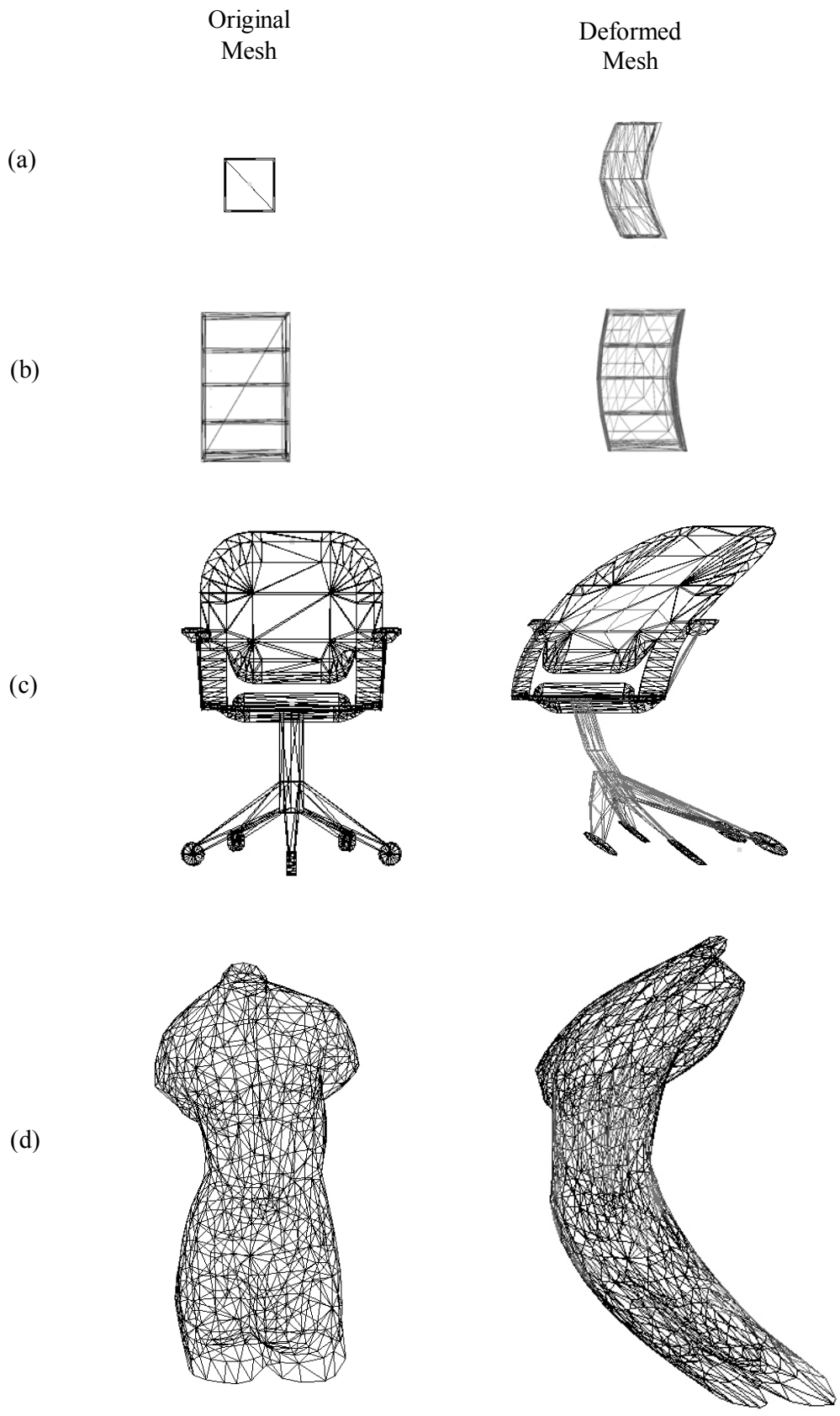

Plate 1: Examples of (a) Cube (b) Bookcase (c) Chair (d) Venus model