Voxel Free-Form Deformations

Ben Kenwright

***Abstract*—A straightforward and efficient deformation algorithm is an important tool for creating more engaging and interactive virtual environments. This paper explores computational factors and algorithms necessary for creating a visually pleasing soft-body deformation effect. We compare the different techniques available, while examining and evaluating the visual and computational trade-offs each method offers. With this in mind, we demonstrate a level of detail subdivision method based upon a grid-spatial partitioning optimisation (voxels and tetrahedrons). We investigate computational speed-ups using the graphical processing units interoperability feature. Having said that, the object voxels, control points, and the associated deformations provide a scalable solution that is suitable for real-time systems. All things considered, we conclude with a discussion on the significance of our work in virtual environments and possible future areas of investigation.**

Keywords–deformation, convex hulls, 3d, convex, video games, real-time computer generated, interactive

I INTRODUCTION

Motivation Geometry in today's virtual worlds, such as, video games and training simulations, tend to be static. It is common to see animated characters composed of mesh hierarchies moving relative to each other, but it is rare to see the individual mesh elements animated (i.e., vertices). Animating hierarchies of interconnected meshes is the traditional approach [1] for portraying an object's movement, but does little to portray the organic non-rigid situation or characteristics of a mesh. In fields other than computer graphics, animators traditionally exaggerate parts of a mesh (for instance, a character's movement) to convey expressions. In interactive environment it is difficult for animation solutions to capture these expressive motions, such as, exaggeration. This is usually because an object generally remains the same shape throughout the simulation.

Soft-Body Mechanics The process of deforming vertices within a mesh lies within the domain of soft-body mechanics. Soft-body mechanics in combination with traditional animation techniques greatly enhances the realism of an object or scene. *Until recently, this process was simply too expensive to consider in real-time environments.* However, with the advancement of computational power, such as, the graphical processing unit (GPU) and massively parallel processing, soft-object animations are becoming a viable real-time solution [5], [16].

High Poly Graphical Meshes The trend for realism and the advancement of computational power, means graphical models, have become more detailed and complex, hundreds of thousands of vertices. The deformation of an object occurs by moving the vertices of the graphical mesh. Typically, due to the high poly count - groups of vertices are attached to control points that artists can move using either pre-recorded animations or parametric curves to attain the desired effect. A crucial factor is that the mesh object has sufficient number of vertices/face. As if the polygon resolution is low, the deformations give rise to a degradation in silhouette edge aliasing.

Contribution The key contributions of this paper are: (1) we evaluate and explore optimisations of the traditional FFD system for real-time environments; (2) we implement a voxel-based system for targeting active control regions; (3) we explore the interoperability of the GPU for sharing data through common framebuffers for computational speed-ups - whereby we control and rendering the deformation using the massively parallel GPU architecture.

II RELATED WORK

Multi-Discipline The creation and control of deformable meshes is an important component in multiple research areas, such as, engineering material analysis and safety testing. However, we focus on an interactive solutions, such as, video games, since it allows the player to visually experience a more life-like and realistic environment. We review a number of important papers and concepts for solving deformation problems in different contexts (e.g., character animation and material analysis) in this section. Figure 2, presents a visual time-line of key publications in the area, starting with the initial free-form deformation work by Sederberg and Parry [22].

Initially The **free-form Deformation technique is commonly known as FFD**. The technique has been around for some time, but was first documented in a SIGGRAPH paper by Sederburg and Parry in 1986 [22]. This FFD technique is used in numerous commercial modelling and animation packages, and forms the groundwork of our work. The technique is intuitive and straightforward to implement and forms the basis of a number of other techniques (e.g., hierarchy ffd) [6]. A number of important deformation techniques have been published 2 over the years, however, a real-time solution that is flexible and scalable would be useful. Similarly, in commercial circles (e.g., video games and training simulations), we see deformation systems but the methods and techniques are propriety owned and not shared publically. Crucially, the soft-body animation should not consume all the system resources. For real-time applications, only a small amount of the overall computational resource is allocated to the animation, since the application needs to run a number of components, like graphics, artificial intelligence, networking, and game-play features.

Texture Displacement A uncomplicated and efficient method for convex deformations (i.e., surface normal displacement) that is able to produce reasonable results is the use of texture-vertex mapping [19], [9]. The concept is most noted for its application in simulating water ripple effects [7]. The approach writes height information to a texture that is used to deform the graphical mesh vertices (e.g., analogous to physical bump-mapping). Information written to the texture are applied to the object mesh using the graphical processing unit (GPU) to achieve a real-time frame-rates. This approach can also be used to write non-deformable feedback (e.g., scratches and marks to the graphical textures).

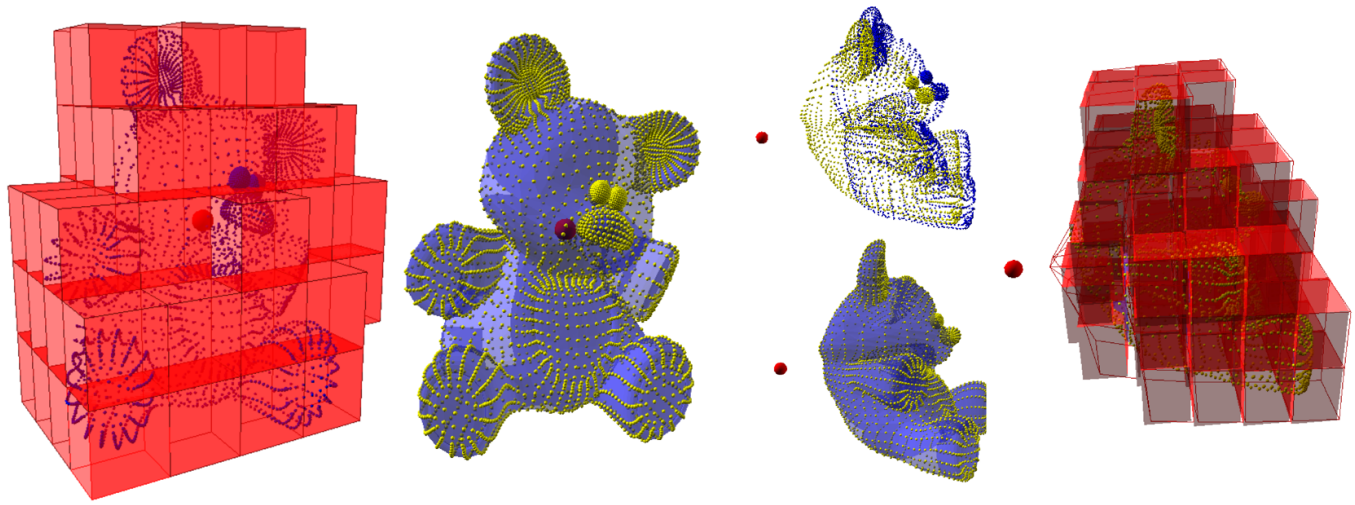

Figure 1. **Bear Model Local Voxel Regions** - Illustrating the influence of linear local 2x2x2 control voxel array. (30240 graphical vertices, 216 control points (5x5x5) - with 85 voxels active or 168 control points). Figure shows a magnetic sphere moving around the object pulling control points in its direction to show squishing and deformation of the underlying graphical mesh.

Finite Element Model This section discusses the representation of the deformation model (i.e., finite element decomposition approximation). The model must be able to handle large deformations and be stable under large time-steps while not hindering the systems performance. Deformation is a geometric measure of strain (e.g., stretching and shearing). Typically, strain models use accurate finite element methods that assume small deformations. These simple linear strain models may cause inflation/expansion issues when the strain cannot separate rotation information [17]. Of course, methods have been developed to attempt to remove the rigid rotation [4], [25]. This solution is popular in interactive applications but the deformation needs to be very small. We use a coarser non-linear finite-element model that is coupled to the high-detailed geometry mesh. This allows us to synthesize physically plausible deformation in real-time, while maintaining a reasonably correct physical model. Our coarser model extracts the key details from the high-detailed graphical mesh, such as, structural inter-connectivity and mass distribution.

Object collision deformations are likely to be large, especially for high speed impacts (e.g., objects hitting a wall or skin deforming when touched). We must take care with these large deformations, as large aesthetically pleasing deformation are difficult to create in some respects (i.e., materials should not look rubbery or jelly-like). In order to handle large deformations, we use a low-dimensional approximation. We express large deformations regardless of object's rigid body centre.

The mesh node positions formulate a finite element decomposition. The strain for each element is computed from the nodal positions in the currently deformed state and the initially non-deformed state. For every element, we determine the influence of each neighbour. As nodes are disturbed from their rest location by external perturbations they influence their neighbouring elements in accordance with their spatial proximity and connectivity strength.

Low-Resolution Control Mesh We explain our low-dimensional control model for deformations. A control mesh technique is used reduce the computational overhead and the mathematical complexity of the model so we can achieve real-time frame-rates. We explain the high-resolution object surface interactions to handle detailed contacts between the object and the environment in an endeavour to realistically mimic the mechanical deformation properties.

Model Reduction & Mesh Embedding The two main techniques for reducing the complexity of a finite element system can be classified into two main types: *modal reduction* and *mesh embedding*. Modal reduction is a popular method for reducing the complexity of a finite element system by using a linear subspace to span a small number of displacement basis vectors to represent the deformation in the body. The eigenmodes obtained from linear modal analysis would be the best basis vectors for small deformation. For large deformation, however, they are not sufficient to capture the non-linear deformation characteristics, so multiple techniques have been suggested to choose a good deformation basis set [2]. Model techniques have successfully been used for real-time solutions, such as, surgery simulators and hand-soft body interaction.

Mesh embedding, which is also called *free-form* deformation [14], uses a low-dimensional coarse volumetric mesh to enclose the entire deformable body in order to represent the behavior of the body. The location of every material point inside the deformable body is determined by interpolating the positions of the neighboring nodes in the mesh. Since the work by Faloutsos et al. [6], mesh embedding techniques have been widely used to simulate soft bodies in the graphics literature [18], [13], [3] We chose mesh embedding to reduce complexity of the deformable body in our simulation system not only because the technique can reduce the model complexity without losing the fine geometry of the object but also because the frame can be manipulated more easily and efficiently using the embedding mesh system compared to modal reduction. In our formulation, the control body elements are considered as an interconnected set of rigid elements that can be solved using iterative penalty-based constraints. The complete system consists of a set of deformable body elements and a rigid body core.

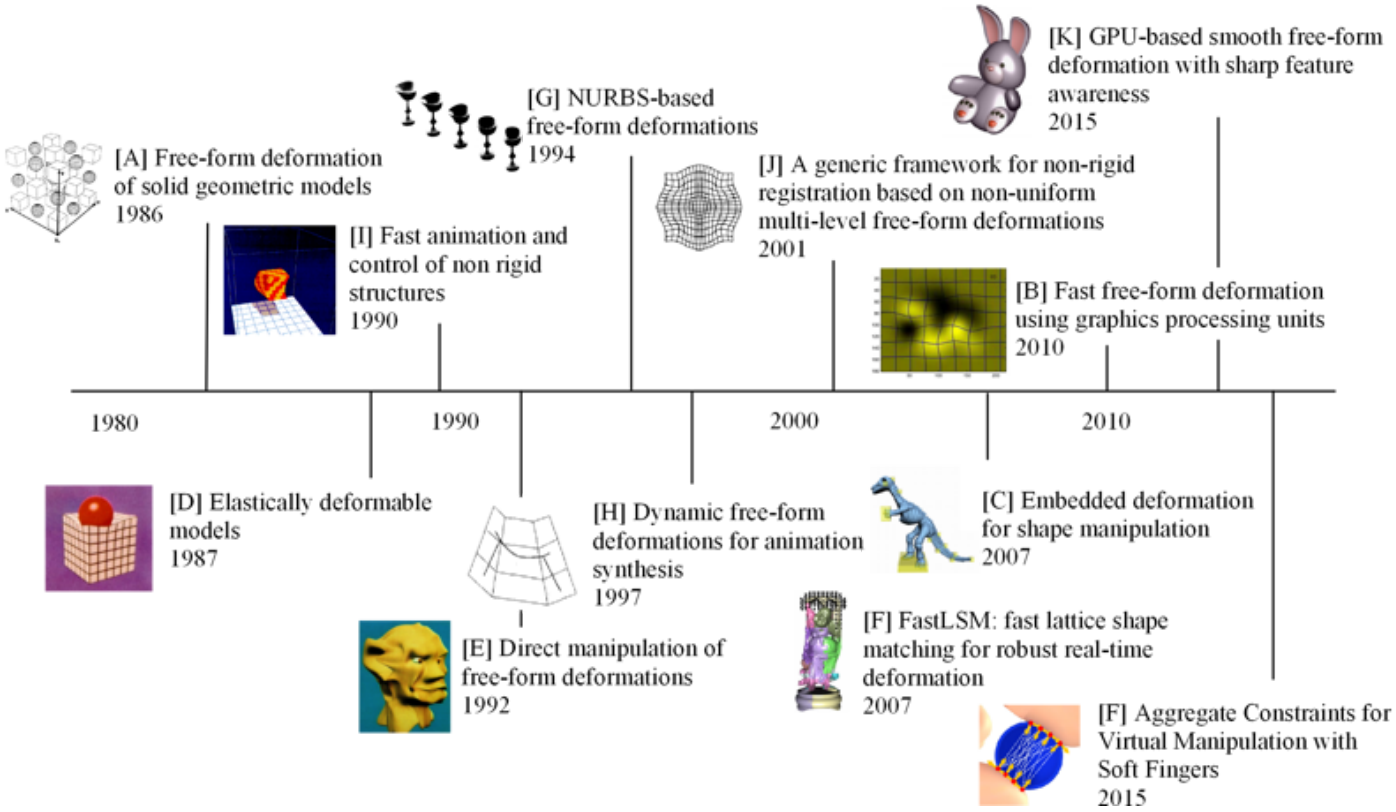

Figure 2. **Timeline** - Graphical timeline showing some of the important soft-body milestones in the area of computer graphics and animation over the past few decades. [A] [22], [B] [16], [C] [23], [D] [25], [E] [8], [F] [20], [G] [15], [H] [6], [I] [26], [J] [21], [K] [5], [L] [24]

III METHOD

Free-Form Deformation (FFD) Algorithm The FFD algorithm comprises three components: (1) transformation of the floating image using the splines and an interpolation function; (2) evaluation of an objective function; (3) and optimisation against this function. Individually, these components may be formulated in a data parallel manner as they mainly consist of 'voxel-wise' computations. However, difficulties associated with computational and memory constraints mean certain aspects are not easily implemented in practice [16].

Cubic B-Splines Interpolation The FFD algorithm consists of locally deforming an image volume using cubic B-Splines. This technique has the desirable feature of **guaranteeing a continuous deformation**. The cubic B-Splines framework is well documented elsewhere, and the details are omitted for brevity. However, a particularly favourable property of the technique is that any deformation produced with a grid of density n can be exactly produced on a grid of density $2n1$. This property has been used in a pyramidal approach in other work implementation [16].

However, **cubic B-Spline methods are extremely computationally expensive**. For this reason in the classical approach only one control point is optimised at a time, which means the whole image does not have to be fully interpolated at each step. The computation of each voxel's position and their new intensities are fully independent and thus their computation is suitable for parallel implementation. Since parallel computation is more efficient when processing large amounts of data concurrently, the optimise all control points and interpolate the whole image at each step.

The technique of Free Form Deformation (FFD) was developed by Parry and Sederberg [22]. The FFD technique embeds an object in a space that is than deformed. The most commonly used **analogy for a FFD is to consider an object embedded in a parallel piped of clear, flexible plastic**. If the lattice structure is deformed, the object inside the lattice will also be deformed.

The lattice structure is composed of Bezier hyperpatches. A hyperpatch is specified by a three-dimensional grid of control points p_{ijk} and defines a volume of space parametrized by the three parameters u,v, and n where $0 < u, v, n < 1$. An FFD block a rectangular volume where each face is a hyperpatch. Let the the three sides be represented by the vectors $(S, TandU)$.

At its heart, FFD is the **metamorphosis of an object from its originally modeled appearance** - known as the object's rest state - to its deformed state. This is accomplished by deforming the object's coordinate system in the following three steps:

- The object to be deformed is embedded in a regular coordinate system defined by three mutually perpendicular axes - the standard (X, Y, Z) axes we are all used to dealing with
- The coordinate system is deformed, allowing its previously straight axes to become curves. Areas of the coordinate system can collapse inwards or expand outwards
- The positions of the object's vertices in the old (regular) coordinate system are updated to match where they ended

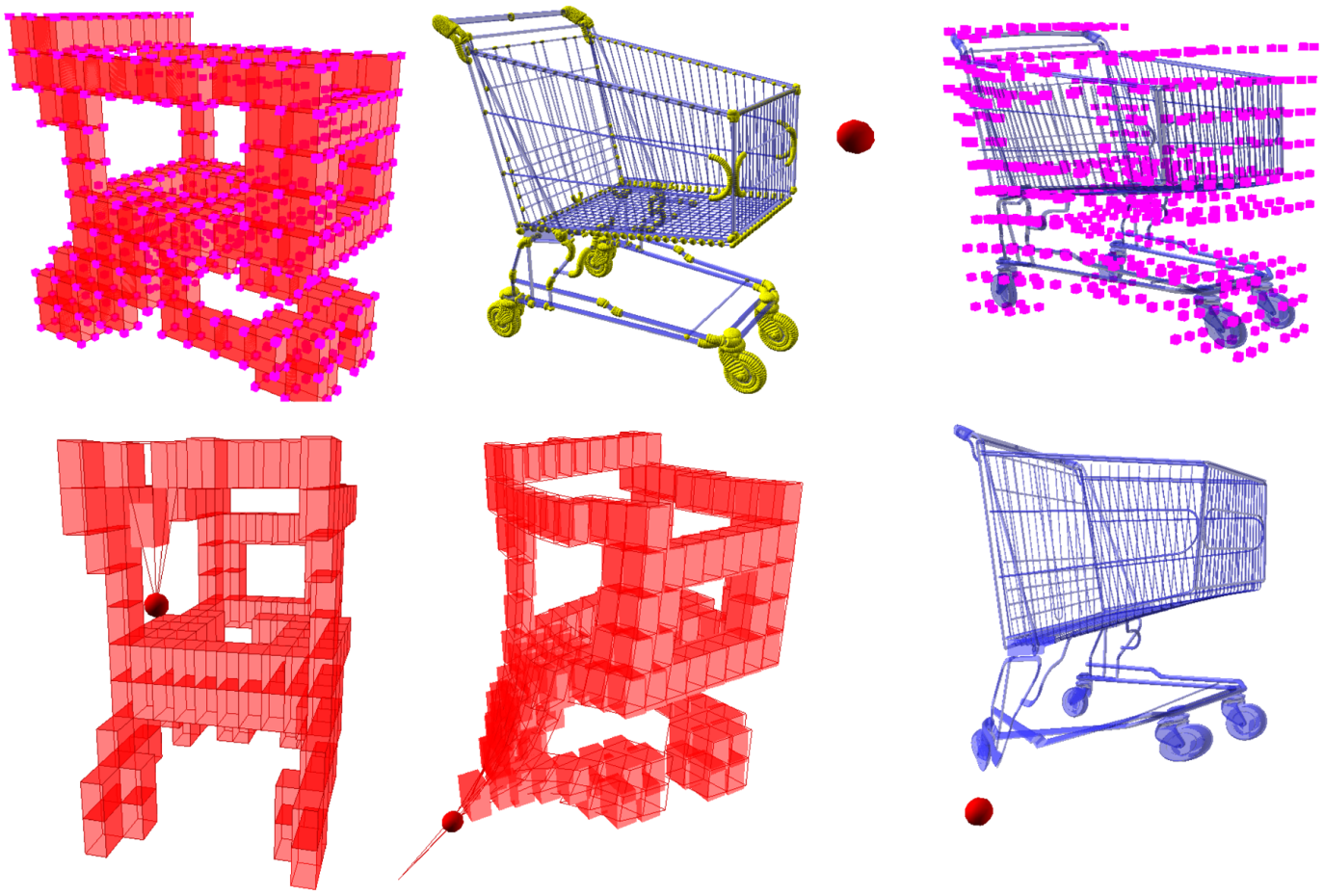

Figure 3. **Trolly Local Voxel Regions** - Illustrating the influence of linear local 2x2x2 control voxel array. (487656 graphical vertices, 1331 control points (10x10x10) - only 188 voxels active or 566 control points).

up after the coordinate system was deformed

Graphical Processing Unit (GPU) The implementation was achieved using CUDA which is an Application Programming Interface developed by NVidia to simplify the interface between CPU (host) and GPU (device) (Figure 4).

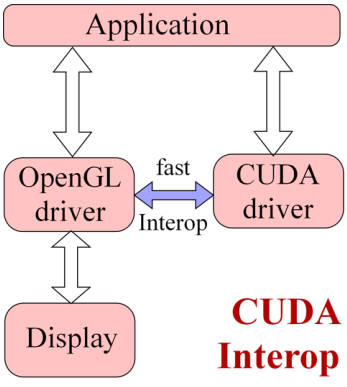

Figure 4. **GPU & CUDA Interop** - Exploiting the massively parallel architecture of the graphical processing unit.

A. *Global Deformation Control*

Every point influences every other point. The control points are calculated:

$$P_{ijk} = X_0 + \frac{i}{l}S + \frac{j}{m}T + \frac{k}{m}U \tag{1}$$

The lattice space is given by: $X(s,t,u) = X_0 + sS + tT + uU$. X0 is the origin of the local coordinate system and S, T, and U lie along the edges of the FFD block. Note that for any point interior to the lattice $0 < s < 1$, $0 < t < 1$, and $0 < u < 1$.

$$\begin{aligned} s &= \frac{T \times U(X - X_0)}{T \times U \cdot S}, \\ t &= \frac{S \times U(X - X_0)}{S \times U \cdot T}, \\ u &= \frac{S \times T(X - X_0)}{S \times T \cdot U}, \end{aligned} \tag{2}$$

$$X_{\text{ffd}} = \sum_{i=0}^{l} \binom{l}{i} (1-s)^{l-i} s^i [\sum_{j=0}^{m} \binom{m}{j} (1-t)^{m-j} t^j \\ [\sum_{k=0}^{n} \binom{n}{k} (1-u)^{n-k} u^t \; P_{ijk}]] \tag{3}$$

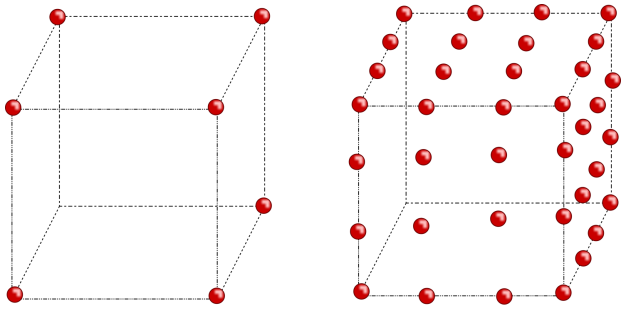

Figure 5. **Local Deformations** - (a) 2x2x2 sub-region and (b) 4x4x4 sub-region.

Global Solution Features

- overall coupled influence - this can be desirable or undesirable in difference circumstances. For example, we may or may not want the effects of a deformation on opposite side of the shape to influence each other
- the model is computationally expensive - since every vertex needs to take into account the influence of 'every' control point. As the number of control points increases the computational cost becomes increasingly expensive
- the method is suitable for exploitation on massively parallel architectures, such as, the GPU [16]
- the deformed mesh may not stay within the control mesh lattice

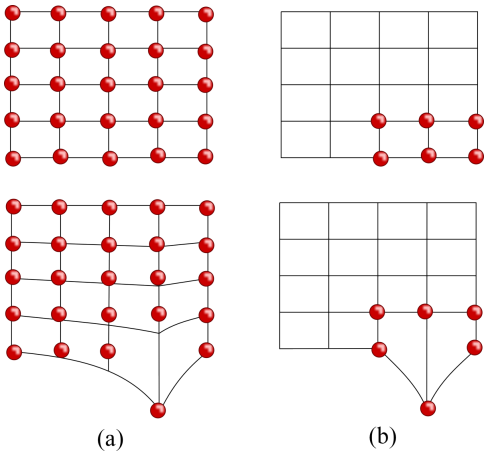

Figure 6. **Global and Local Deformations** - (a) global influence and (b) sub-region.

B. Local Deformation Control

One disadvantage of using a single FFD block to control animation of an object is that you are applying global deformations. A deformation will alter the values of every vertex in the object to a greater or lesser extent. This gives the visual impression that the whole object is one jelly-like mass.

This is undesirable in many situations. For instance, we might be deforming a car as a result of a crash. If the car were to be hit at its front-right corner, we would want that corner of the car to crumple, but the rest of the car should remain unaffected.

We can apply changes more accurately to a particular part of the object by increasing the degree, and hence the number of control points, of the FFD. However, the changes applied are still global, and will still affect every point in the object, although to an increasingly small degree. The computational cost of the deformation is also greatly increased remember that with an FFD block of degree n the inner loop is iterated n3 times per vertex.

The solution is to use more than one FFD block per object. In fact, there are no restrictions on the use of FFDs they can even overlap within an object. We might define the car as being encased in four FFDs, one for each quarter of the car (as seen from above). Now during impact, if the first FFD block is modified, only the affected part of the car will change.

When joining FFDs the same continuity constraints apply as when joining Bezier curves or patches that is, at a minimum, points which meet should be in the same position in space, and preferably should have some level of derivative continuity. This ensures that the object doesnt develop any jagged edges or discontinuities during the deformation.

Cubic FFD Another way to reduce the number of calculations is to reduce the degree of the FFD block. While a cubic FFD requires 64 iterations of the function's main loop per vertex to calculate the deformed positions. There is also the possibility of a quadratic FFD block, represented by a 3x3x3 array of control points for even fewer iterations.

The basic structures of the cubic Bezier volume is given in Figure 5(b). We set all the control points up to form a 4x4x4 regular coordinate grid. After defining Bezier curves and surfaces, the extension to volumes simply involves increasing the number of control points and adding another parameter to the Bezier equation. The array of control points will contains 64 vectors in a 4x4x4 array, and the curve is defined in terms of three parameters given below in Equation 4:

$$\sum_{i=0}^{3} \sum_{j=0}^{3} \sum_{k=0}^{3} B_{i,3}(u) B_{j,3}(v) B_{k,3}(w) P_{ijk} \tag{4}$$

where Equation 4, is known as the Tricubic Bezier Hyperpatch, with $B()$ known as the Berstein polynomial blending function, defined as: $B_{\nu,n}(x) = \binom{n}{\nu} x^{\nu} (1-x)^{n-\nu}, \quad \nu = 0, \ldots, n.$ and $\binom{n}{\nu}$ known as the binomial coefficient. We also provide an expanded implementation of Equation 4 in Support Material Listing.

Linear FFD Linear FFDs, defined by a 2x2x2 grid of points can be used to quickly apply shear or taper to objects, see Figure 5(a).

Local Solution Features

- only neighbouring control points influence the deformation. We can modify the distance of influence (e.g., 2x2x2 or 4x4x4 regions)
- the model is much less expensive compared to the global solution - especially for large numbers of control points
- the method is suitable for exploitation on massively parallel architectures, such as, the GPU [16]

C. Global or Local Deformation

There are many situations where a technique for deforming meshes could be applied in interactive environments:

1) In cartoon-like animation, it can be used to add personality to a mesh's animation sequence. For instance, a character that is jumping will squat down first, and an angry character might pulse with rage. This is an example of a **global deformation** the whole object is controlled by one deformation.
2) During a crash in a car racing game, the area where the impact occurred could be deformed, and remain so afterwards. This gives a convincing look of damage and saves the need for having multiple models of each car. This is an example of **local deformation** we do not want the whole car to deform just because the front bumper is bent.
3) A hierarchy of deformations can animate characters with more convincing muscle tone than hierarchies of objects are capable of portraying. This is an example of **hierarchical deformation**, and the deformations would overlap. For instance, at the joints of a bone how the flesh looked would be dependent on the position of both bones.

D. Voxels & Optimisation

A **pixel** (picture element) defines a point in two dimensional space with its x and y coordinates; while a **voxel** is a unit of information defining a point in three-dimensional space (i.e., the combination of the words volumetric and pixel to represent a volume element). In three-dimensional space, each of the coordinates is defined in terms of its position, color, and density. For example, a cube may represent any region of space expressed by a x, y and z coordinate and its size. This information enables us to **discretize** a three-dimensional world into elements.

A graphical mesh is encapsulated by a bounding box. The bounding box is subdivides into voxels. Each voxel is made up of eight corners which represent the control points Each mesh vertex within with the FFD is influenced by one or more control points and hence voxels. For 'local' FFD implementations, we optimise the solution by culling voxels (and control points) that have no influence over the mesh deformation (e.g., see Figure 7).

E. Tetrahedron

We are able to apply the FFD principle to tetrahedrons [10]. Defining the four corners of the tetrahedron (ABCD) and the point within (X) as shown in Figure 8. The interpolation function for tetrahedron cell is a linear one of the type in Equation 1. The four coefficients are calculated the same way as the axis aligned box. The natural coordinates are again taken

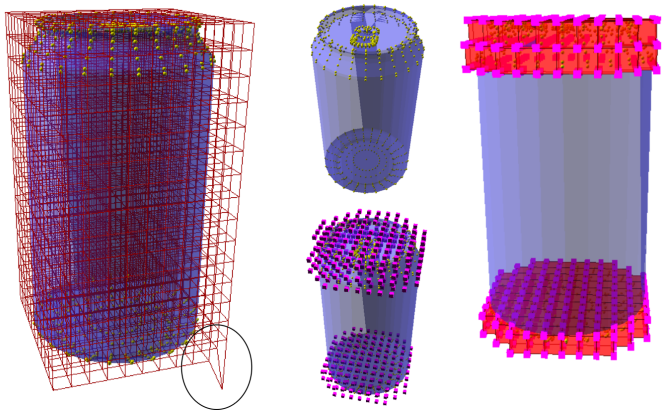

Figure 7. **Drinks Can Local Voxel Regions** - Illustrating the influence of linear local 2x2x2 control voxel array. (4720 graphical vertices, 1936 control points (10x15x20) - only 190 voxels active or 490 control points).

to vary from 0 to 1 in the non-dimensional cell. With reference to Figure 8, Equation 5 calculates the weight coordinates. Our vector formulation also correlates with the work of Kenwright and Lane [12] who solved a system of equations by inverting a 3x3 matrix.

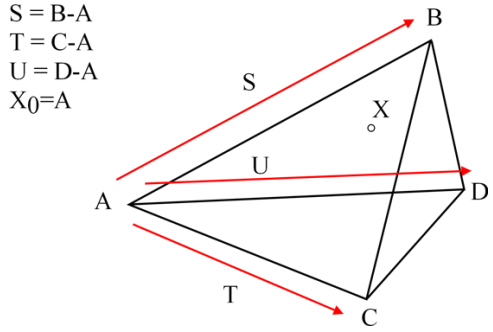

Figure 8. **Tetrahedron** - Barycentric coordinates to calculate the deformation weights.

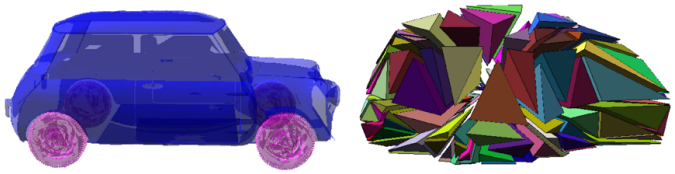

Figure 9. **Simple Object Tetrahedron Decomposition** - Forming a convex hull with the centroid as a reference point a control mesh formed of tetrahedrons is easily formed compared to using cuboid voxels (e.g., see Figure 11).

$$
\begin{aligned}
s &= \frac{T \times U(X - X_0)}{T \times U \cdot S}, \\
t &= \frac{S \times U(X - X_0)}{S \times U \cdot T}, \\
u &= \frac{S \times T(X - X_0)}{S \times T \cdot U}, \\
X &= X_0 + sS + tT + uU
\end{aligned}
\tag{5}
$$

where s, t, and u are the scalar weights for the point within the tetrahedron. The scalar weights are calculated initially at the start (i.e., before any deformation takes place). As the control

points (corner vertices of the tetrahedron) move, the morphed vertex X is recalculated.

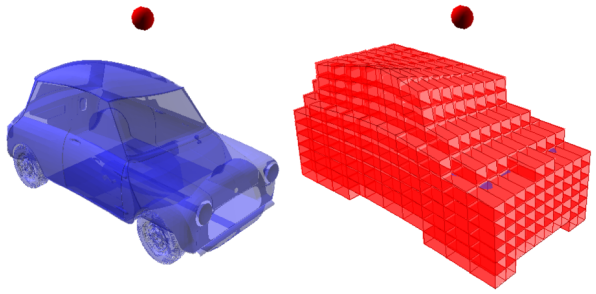

Figure 10. **Vehicle Local Voxel Regions** - Illustrating the influence of linear local $2x2x2$ control voxel array. (327141 graphical vertices, 1331 control points ($10x10x10$) - only 517 voxels active or 953 control points). Circled a control point that does not influence any mesh points when moved.

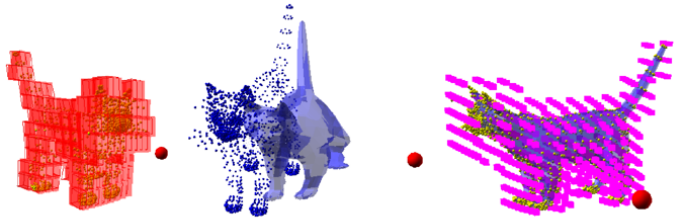

Figure 11. **Cat Model Local Voxel Regions** - Illustrating the influence of linear local $2x2x2$ control voxel array. (11862 graphical vertices, 1331 control points ($10x10x10$) - only 280 voxels active or 579 control points).

IV EXPERIMENTAL RESULTS

The simulations with all the test models were implemented on a desktop machine with 3.2 GHz Intel i7 CPU and NVIDIA GeForce GTX 480 GPU. The number of control points allowed greater fine detailed influence over the deformation. Additionally, reducing the regional influence (i.e., from global to local) affects the computational cost and visual appearance.

Scalability We endeavour to automate the deformation process rather than depending on artist intervention for modelling the underlying low-poly control mesh. This enables us to adapt the detail of the transformation for different target audiences (i.e., reduce the model complexity to more coarser representations for environments with limited resources, such as, memory and processing power).

V CONCLUSION

A straightforward and efficient soft-body algorithm is an important tool for creating more engaging, interactive, and life-like scenes. The method we have presented allows the solution to be customized to the systems needs (e.g., scalable, efficient, and automatic through artistic customisation). The flexibility of the approach within this paper allows developers and artists to design more attractive solutions that capture the imagination without sacrificing resources. Overall, we focused on a low-dimensional voxel-based model. The method can be used off-line to create pre-canned animations for animated files or in real-time, while supporting a diverse set of characteristics.

The soft-body solution in this paper for deforming object vertices is practical. We provide a set of supporting code samples to support the paper. FFDs are particularly useful when combined with objects represented by Bezier patches, however, the solution is flexible enough to be combined with physics-based control systems, such as, constraint solvers and penalty-based springs to create a dynamic solution [11].

ACKNOWLEDGEMENTS

A special thanks to reviewers for taking time to review this article and provide insightful comments and suggestions to help to improve the quality of this article.

References

[1] Ilya Baran and Jovan Popović. Automatic rigging and animation of 3d characters. In *ACM Transactions on Graphics (TOG)*, volume 26, page 72. ACM, 2007. 1

[2] Jernej Barbič and Doug L James. Real-time subspace integration for st. venant-kirchhoff deformable models. In *ACM Transactions on Graphics (TOG)*, volume 24, pages 982–990. ACM, 2005. 2

[3] Steve Capell, Matthew Burkhart, Brian Curless, Tom Duchamp, and Zoran Popović. Physically based rigging for deformable characters. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 301–310. ACM, 2005. 2

[4] Steve Capell, Seth Green, Brian Curless, Tom Duchamp, and Zoran Popović. A multiresolution framework for dynamic deformations. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 41–47. ACM, 2002. 2

[5] Yuanmin Cui and Jieqing Feng. Gpu-based smooth free-form deformation with sharp feature awareness. *Computer Aided Geometric Design*, 35:69–81, 2015. 1, 3

[6] Petros Faloutsos, Michiel Van De Panne, and Demetri Terzopoulos. Dynamic free-form deformations for animation synthesis. *Visualization and Computer Graphics, IEEE Transactions on*, 3(3):201–214, 1997. 1, 2, 3

[7] Alain Fournier and William T Reeves. A simple model of ocean waves. In *ACM Siggraph Computer Graphics*, volume 20, pages 75–84. ACM, 1986. 1

[8] William M Hsu, John F Hughes, and Henry Kaufman. Direct manipulation of free-form deformations. In *ACM Siggraph Computer Graphics*, volume 26, pages 177–184. ACM, 1992. 3

[9] Matthew Johnson-Roberson, Oscar Pizarro, Stefan B Williams, and Ian Mahon. Generation and visualization of large-scale three-dimensional reconstructions from underwater robotic surveys. *Journal of Field Robotics*, 27(1):21–51, 2010. 1

[10] Ben Kenwright. Free-form tetrahedron deformation. In *International Symposium on Visual Computing*, pages 787–796. Springer, 2015. 6

[11] Ben Kenwright and Graham Morgan. Practical introduction to rigid body linear complementary problem (lcp) constraint solvers. *Algorithmic and Architectural Gaming Design*, pages 159–205, 2012. 7

[12] David N Kenwright and David A Lane. Optimization of time-dependent particle tracing using tetrahedral decomposition. In *Proceedings of the 6th conference on Visualization'95*, page 321. IEEE Computer Society, 1995. 6

[13] Lily Kharevych, Patrick Mullen, Houman Owhadi, and Mathieu Desbrun. Numerical coarsening of inhomogeneous elastic materials. In *ACM Transactions on Graphics (TOG)*, volume 28, page 51. ACM, 2009. 2

[14] Junggon Kim and Nancy S Pollard. Fast simulation of skeleton-driven deformable body characters. *ACM Transactions on Graphics (TOG)*, 30(5):121, 2011. 2

[15] Henry J Lamousin and Warren N Waggenspack Jr. Nurbs-based free-form deformations. *Computer Graphics and Applications, IEEE*, 14(6):59–65, 1994. 3

[16] Marc Modat, Gerard R Ridgway, Zeike A Taylor, Manja Lehmann, Josephine Barnes, David J Hawkes, Nick C Fox, and Sébastien Ourselin. Fast free-form deformation using graphics processing units. *Computer methods and programs in biomedicine*, 98(3):278–284, 2010. 1, 3, 5, 6

[17] Matthias Müller and Markus Gross. Interactive virtual materials. In *Proceedings of Graphics Interface 2004*, pages 239–246. Canadian

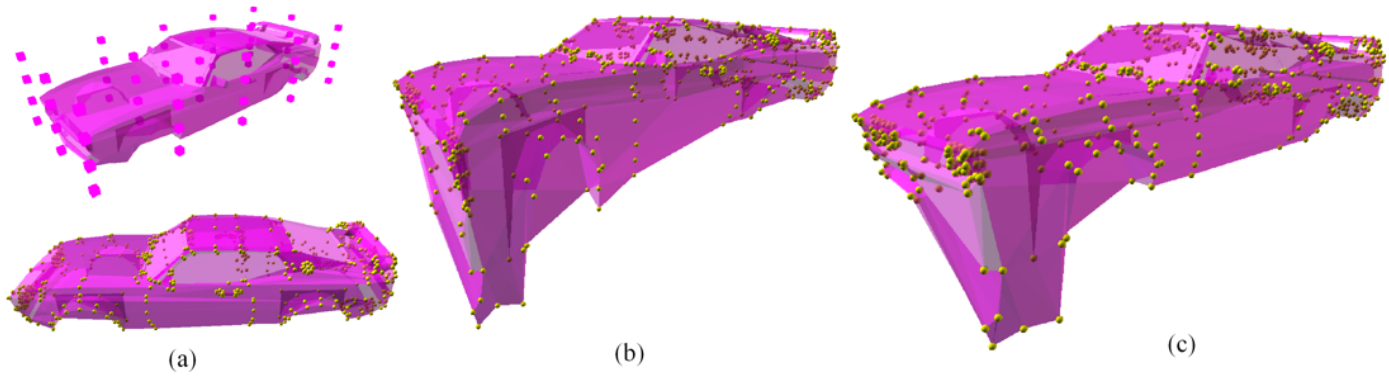

Figure 12. **Global & Local Simulation** - Deformation example (a) initial model mesh and control points, (b) global and (c) local deformation influences. Graphical mesh with 3810 vertices and 60 control points (i.e., 3x4x5 distribution along x, y, and z axis).

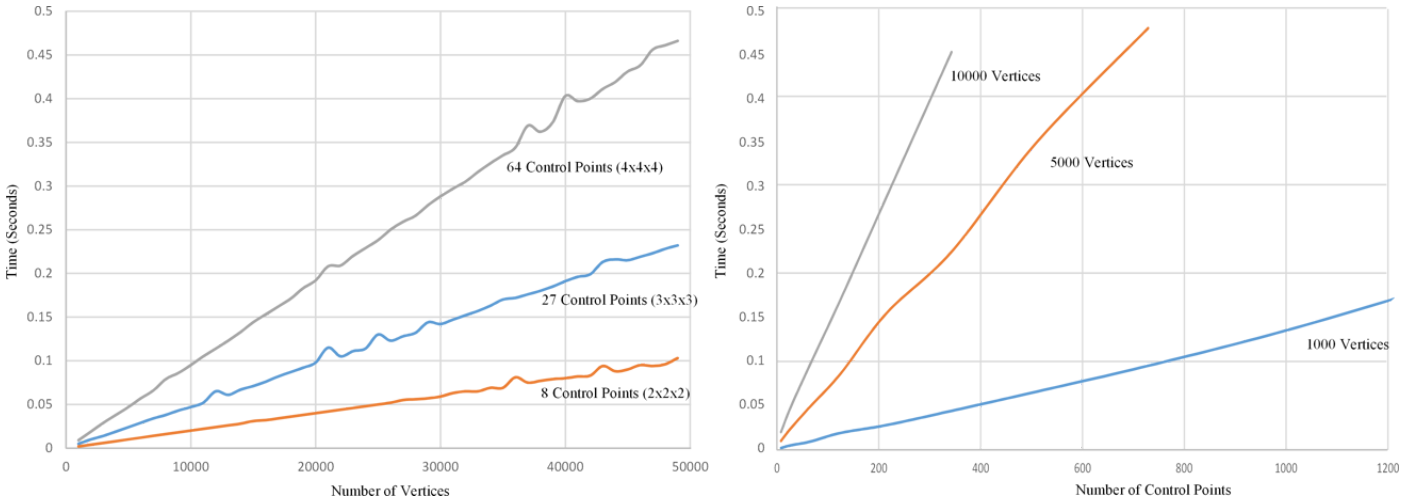

Figure 13. **Global Simulation Timing** - Timing information for varied numbers of control points and graphical vertices. (a) Number of Vertices vs Time (Fixed Number of Control Points), and (b) Number of Control Points vs Time (Fixed Number of Vertices).

Human-Computer Communications Society, 2004. 2

[18] Matthieu Nesme, Paul G Kry, Lenka Jeřábková, and François Faure. Preserving topology and elasticity for embedded deformable models. In *ACM Transactions on Graphics (TOG)*, volume 28, page 52. ACM, 2009. 2

[19] Masaaki Oka, Kyoya Tsutsui, Akio Ohba, Yoshitaka Kurauchi, and Takashi Tago. Real-time manipulation of texture-mapped surfaces. In *ACM SIGGRAPH Computer Graphics*, volume 21, pages 181–188. ACM, 1987. 1

[20] Alec R Rivers and Doug L James. Fastlsm: fast lattice shape matching for robust real-time deformation. *ACM Transactions on Graphics (TOG)*, 26(3):82, 2007. 3

[21] Julia A Schnabel, Daniel Rueckert, Marcel Quist, Jane M Blackall, Andy D Castellano-Smith, Thomas Hartkens, Graeme P Penney, Walter A Hall, Haiying Liu, Charles L Truwit, et al. A generic framework for non-rigid registration based on non-uniform multi-level free-form deformations. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2001*, pages 573–581. Springer, 2001. 3

[22] Thomas W Sederberg and Scott R Parry. Free-form deformation of solid geometric models. In *ACM SIGGRAPH computer graphics*, volume 20, pages 151–160. ACM, 1986. 1, 3

[23] Robert W Sumner, Johannes Schmid, and Mark Pauly. Embedded deformation for shape manipulation. *ACM Transactions on Graphics (TOG)*, 26(3):80, 2007. 3

[24] Anthony Talvas, Maud Marchal, Christian Duriez, Miguel Otaduy, et al. Aggregate constraints for virtual manipulation with soft fingers. *Visualization and Computer Graphics, IEEE Transactions on*, 21(4):452–461, 2015. 3

[25] Demetri Terzopoulos, John Platt, Alan Barr, and Kurt Fleischer. Elastically deformable models. In *ACM Siggraph Computer Graphics*, volume 21, pages 205–214. ACM, 1987. 2, 3

[26] Andrew Witkin and William Welch. Fast animation and control of nonrigid structures. In *Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 243–252. ACM, 1990. 3

Appendix

Listing 1. Blended position given the 8 control points for the vertex within the cube and the pre-calculated weights.

```
// Local deformation 2x2x2 Region
Vector3 WeightLinearCube2x2x2(
const vector<Vector3>& controlpoints,
float s, float t, float u)
{
// 8 corners of a cube
DBG_ASSERT(controlpoints.size()==8);

DBG_ASSERT (s >= 0.0f && s <= 1.0f);
DBG_ASSERT (t >= 0.0f && t <= 1.0f);
DBG_ASSERT (u >= 0.0f && u <= 1.0f);

float BU[2], BV[2], BW[2];

BU[0] = (1-s);  BU[1] = s;
BV[0] = (1-t);  BV[1] = t;
BW[0] = (1-u);  BW[1] = u;

Vector3 ret(0,0,0);
int n = 0;
for (int i = 0; i < 2; i++)
for (int j = 0; j < 2; j++)
```

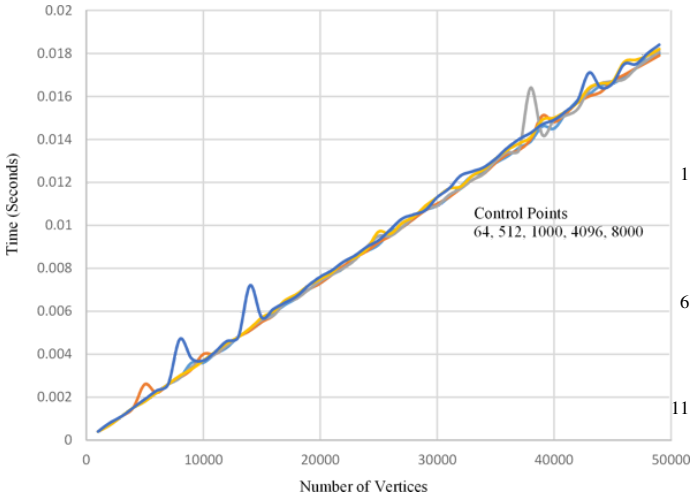

Figure 14. **Linear Simulation Timing** - Timing information for varied numbers of control points and graphical vertices (2x2x2 sub control region). Number of Vertices vs Time (Fixed Number of Control Points). Due to the linear relationship between control points and groups of graphical vertices we are able to create large numbers of control points and have large numbers of mesh vertices (compare with the global timing in Figure 13). Note - the bumps in the graph are due to cache hits and other system resources.

```
for (int k = 0; k < 2; k++)
{
ret += controlpoints[n] * (BU[i] * BV[j] * BW[k]);
n++;
}
return ret;
}// End WeightLinearCube2x2x2(..)
```

Listing 2. Blended position given the 64 control points for the vertex within the cube-lattice and the pre-calculated weights.

```
// Local deformation 4x4x4
Vector3 WeightCubeVolume4x4x4(
const vector<Vector3>& controlpoints,
float s, float t, float u)
{
// 4x4x4 points that make up the cube region
DBG_ASSERT(controlpoints.size()==64);

DBG_ASSERT (s >= 0.0f && s <= 1.0f);
DBG_ASSERT (t >= 0.0f && t <= 1.0f);
DBG_ASSERT (u >= 0.0f && u <= 1.0f);

float BU[4], BV[4], BW[4];

// X Bezier Basis Functions
BU[0] = (1.0f - s) * (1.0f - s) * (1.0f - s);
BU[1] = 3.0f * s * (1 - s) * (1.0f - s);
BU[2] = 3.0f * s * s * (1.0f - s);
BU[3] = s * s * s;

// Y Bezier Basis Functions
BV[0] = (1.0f - t) * (1.0f - t) * (1.0f - t);
BV[1] = 3.0f * t * (1 - t) * (1.0f - t);
BV[2] = 3.0f * t * t * (1.0f - t);
BV[3] = t * t * t;

// Z Bezier Basis Functions
BW[0] = (1.0f - u) * (1.0f - u) * (1.0f - u);
BW[1] = 3.0f * u * (1 - u) * (1.0f - u);
BW[2] = 3.0f * u * u * (1.0f - u);
BW[3] = u * u * u;

Vector3 ret(0,0,0);
int n = 0;
for (int i = 0; i < 4; i++)
for (int j = 0; j < 4; j++)
for (int k = 0; k < 4; k++)
{
ret += controlpoints[n] * (BU[i] * BV[j] * BW[k]);
n++;
}
return ret;

}// End WeightCubeVolume4x4x4(..)
```

Listing 3. Tetrahedron calculate the three weights, given the tetrahedron corners and the vertex (within) the tetrahedron volume.

```
void
CalculateTetrahedronWeight(const vector<Vector3>&
     tetrahedronCorners,
const Vector3& point,
float& s, float& t, float& u)
{
DBG_ASSERT(tetrahedronCorners.size()==4);

const Vector3& A = tetrahedronCorners[0];
const Vector3& B = tetrahedronCorners[1];
const Vector3& C = tetrahedronCorners[2];
const Vector3& D = tetrahedronCorners[3];

const Vector3& X = point;

Vector3 S = B-A;
Vector3 T = C-A;
Vector3 U = D-A;

s = Dot( Cross(T, U), X-A ) / Dot( Cross( T, U ), S );

t = Dot( Cross(S, U), X-A ) / Dot( Cross( S, U ), T );

u = Dot( Cross(S, T), X-A ) / Dot( Cross( S, T ), U );

DBG_ASSERT(s>=0 && s<=1.0f);
DBG_ASSERT(t>=0 && t<=1.0f);
DBG_ASSERT(u>=0 && u<=1.0f);
}// CalculateTetrahedronWeight(..)
```

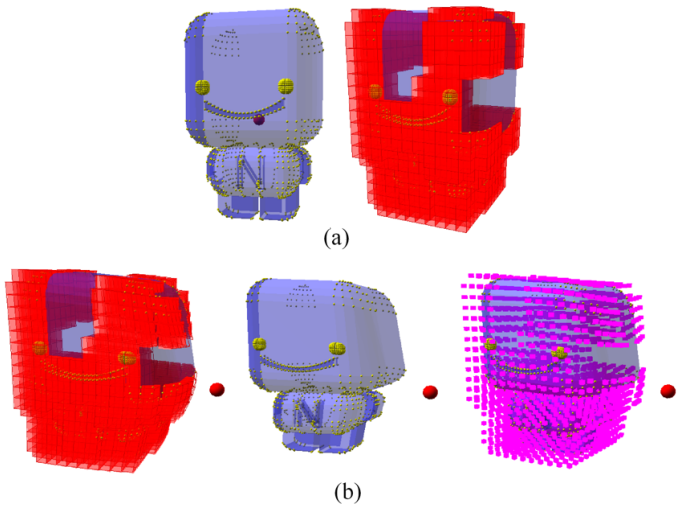

Figure 15. **Local Simulation 4x4x4** - Timing information for a 4x4x4 local grid. Graphical mesh 8058 vertices and the control mesh 12x18x12 points (3211) - optimised to 2084 active voxels for 4x4x4 local FFD. (a) Original mesh and (b) mesh deformed and control points.