Beyond Volume Rendering: Visualization, Haptic Exploration, and Physical Modeling of Voxel-based Objects

Sarah F. Gibson

TR95-04   December 1995

Abstract

For some data sources, a voxel-based representation can provide much more informative data visualization than the surface-based representation of conventional computer graphics. For example, the image data from a 3D Magnetic Resonance Image (MRI) scan consists of a wealth of information about internal structure, function, and anatomy. Using volume rendering with appropriate preprocessing, all of this information can be effectively presented to the attending physician or radiologist. In contrast, surface models derived from the MRI data discard internal structure, cannot be used to represent objects with poorly defined edges or surfaces, and are sensitive to segmentation errors. In this paper, we propose the use of a voxel-based data representation not only for visualization, but also for modeling objects and structures derived from volumetric data. The paper describes work in progress towards demonstrating the utility of a voxel-based format for modeling physical interactions between virtual objects. Data structures are presented that help to optimize storage requirements and preserve object integrity during object movement. An adaptation to volume rendering algorithms is discussed that enables objects to be rendered individually and then combined in a final compositing step. Finally, algorithms and prototype systems are presented that use a voxel-based format to model physical interactions between objects. These physical interactions include collision detection and avoidance of object interpenetration, haptic, or tactile, exploration of virtual objects using a force feedback device, and object deformation.

Revision History:—

1. First version: January 31, 1995

Introduction

For some data sources, a voxel-based representation can provide much more informative data visualization than the surface-based representation of conventional computer graphics. For example, the image data from a 3D Magnetic Resonance Image (MRI) scan consists of a wealth of information about internal structure, function, and anatomy. Using volume rendering with appropriate preprocessing, all of this information can be effectively presented to the attending physician or radiologist. In contrast, surface models derived from the MRI data discard internal structure, cannot be used to represent objects with poorly defined edges or surfaces, and are sensitive to segmentation errors.

In this paper, we propose the use of a voxel-based data representation not only for visualization, but also for modeling objects and structures derived from volumetric data. The paper describes work in progress towards demonstrating the utility of a voxel-based format for modeling physical interactions between virtual objects. Data structures are presented that help to optimize storage requirements and preserve object integrity during object movement. An adaptation to volume rendering algorithms is discussed that enables objects to be rendered individually and then combined in a final compositing step. Finally, algorithms and prototype systems are presented that use a voxel-based format to model physical interactions between objects. These physical interactions include collision detection and avoidance of object interpenetration, haptic, or tactile, exploration of virtual objects using a force feedback device, and object deformation.

Introduction

New medical imaging technologies such as MRI or Computed Tomography (CT) can provide high quality, high resolution images. When stored in a voxel-based format, these data can provide a wealth of information about a patient's internal anatomy. Volume rendering of the voxel-based data is an effective way to present this 3D data to the attending physician or radiologist.

Recent trends have provided a growing interest in using these 3D images not only to visualize anatomy, but also to generate patient-specific anatomical models for pre-surgical planning and surgical simulation. Such models have been used for implant design in orthopedics, for planning a surgical approach, for training and education, and for predicting the outcome of a planned surgery (for example, see papers in [DIGIO94] and [ROBB94]).

Unfortunately, surface-based models created from the 3D image data discard interior structure and are prone to estimation errors, especially where details in the surface (such as small fractures in a bone surface) are small relative to the resolution of the data set. Thus, just as in image visualization, for some data sources a voxel-based data format which preserves interior structure and objects that do not have well defined surfaces may be more appropriate than surface-based modeling methods.

In order to perform pre-surgical planning and surgical simulation on a voxel-based anatomical model, algorithms must be capable of modeling the movement and interaction among several objects in the scene. In most cases, objects must be prevented from interpenetrating as they move about. This requires that collisions between objects must be detected and avoided. When objects contact each other, they should respond in a physically realistic way, which means that we must be able to model the transfer of energy between objects and the shape changes of deformable objects. This paper discusses several algorithms and prototype systems that address these requirements.

Background

New imaging and data sampling technologies have resulted in large, multi-dimensional arrays of data. Conventional sources of volumetric data include: imaging technologies, such as magnetic resonance imaging (MRI) or computed tomography (CT); spatial data sampling, such as geological core sampling or the measurement of temperature or pressure by weather balloons for climate studies; and numerical simulations such as 4D studies of fluid flow or turbulence. More recent sources of volumetric data include industrial CT, finite-element analysis techniques, and "voxelization" methods used to convert CAD and CSG models into voxel-based data representations that can be visualized with voxel-based rendering methods [KAUF87].

Early methods to visualize volumetric data used surface rendering techniques to display polygon-based surface models derived from the volumetric data. Since the late 1970's, volume rendering techniques have been developed which enable more direct visualization of the volumetric data. With appropriate preprocessing, volume rendering can be used to visualize surfaces, interior structure, and objects that do not have well defined surfaces. Many effective volume editing tools, volume rendering algorithms and data compression schemes have been developed (see for example [KAUF90A]). Although processing needs and large memory requirements are still major hurdles that need to be overcome, faster algorithms and special-purpose hardware are enabling real-time volume rendering of data of significant size and resolution [NEUM93B][CABRAL94].

Recently, Kaufman et al [KAUF93] have introduced the field of volume graphics, where a voxel-based data format is used to represent graphical objects that are customarily represented by surface-based models. They have demonstrated that many of the graphical effects, such as shading and reflectance, that are available in surface-based graphics representation are also possible using volume graphics [KAUF88], [AVILA92], [YAGEL92]. Assuming that memory needs and processing requirements can be met effectively, Kaufman et al assert that volume graphics has the potential to supersede surface-based graphics just as 2D raster graphics superseded vector graphics. Whether or not this potential is realized will depend on many factors. However, as occurs in data visualization, some objects will be more accurately modeled using a voxel-based volume graphics format than conventional graphics formats.

Data Structures

There are two important considerations when designing data structures for voxel-based objects that interact in a virtual environment. First, the data structure of the virtual space must be simple and easily updated. While many data structures have been devised to reduce memory requirements of voxel-based data, these generally require significant preprocessing and are not suitable for an environment in which objects can continuously move and change shape. The second consideration is that each voxel can consist of many elements. For example, in an image volume, each voxel may contain color, opacity, a gradient vector, reflectance parameters, material strength and other properties. Storing these lengthy voxel data structures directly in the large array of the virtual environment is very inefficient if there is a lot of empty space between objects. In addition, if object voxels were moved about the discrete array structure of the virtual environment, they would have to be resampled for non-integer movements. Accumulated sampling error would quickly lead to complete degradation of an arbitrarily moving object.

Figure 1 illustrates the data structures used by algorithms and prototype systems discussed in this paper. Two separate data structures are used to minimize storage requirements and maintain the integrity of complex, moving objects while providing an easily updated structure for the virtual environment. An occupancy map with dimensions of the virtual environment is used to monitor interactions among objects and global changes in the environment. Each element of the occupancy map is large enough for a single address and contains either a null pointer or an address of an object voxel data structure. Voxel data structures consisting of elements such as color, opacity, reflectance coefficients, vector gradient, material properties, position, neighbor addresses, etc. are stored in compact object arrays. The occupancy map is initialized by mapping the addresses of voxels in each object into the occupancy map using the voxel positions in object space coordinates and a mapping transformation from object space to the virtual environment space.

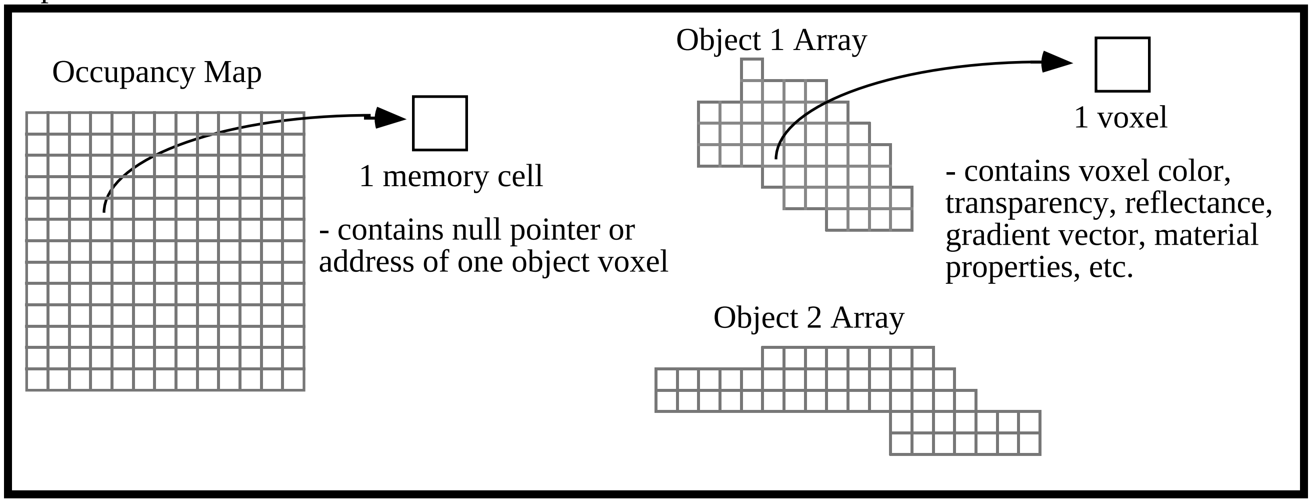
Occupancy Map
1 memory cell
- contains null pointer or address of one object voxel
Object 1 Array
1 voxel
- contains voxel color, transparency, reflectance, gradient vector, material properties, etc.
Object 2 Array

Figure 1. Data structures for a voxel-based object representation. The occupancy map consists of regularly spaced memory cells of integer size that contain either a null pointer or an address of one object voxel. The occupancy map is the size of the application's virtual world. The object arrays consist of data structures each containing the information for a single voxel. This voxel data structure can include many properties and attributes including voxel color, transparency, gradient vector, position vector, material properties, etc. Object arrays are not necessarily rectangular; object voxels are not necessarily evenly spaced or of equal sizes; and voxels are not necessarily ordered within the array. Since a voxel-based graphics system is very memory intensive, the purpose of this double data structure is to pack the large voxel structures of the objects as efficiently as possible while maintaining a map of the entire virtual space for modeling interaction between objects.

Object-based Volume Rendering

Volume rendering of large data volumes is complicated by the fact that voxels must be composited in order, either from the front of the data volume (facing the viewing plane) to the back of the object or from back-to-front. Fortunately, if a large data volume is segmented into non-overlapping convex sub-volumes, 2D intermediate images consisting of accumulated color and transparency through the sub-volume can be calculated and composited to form the volume rendered image of the entire volume. This method is illustrated in Figure 2. It has been used in several algorithms that distribute volume data over multiple processors for parallel rendering [HSU93], [MA93], [NEUM93A].

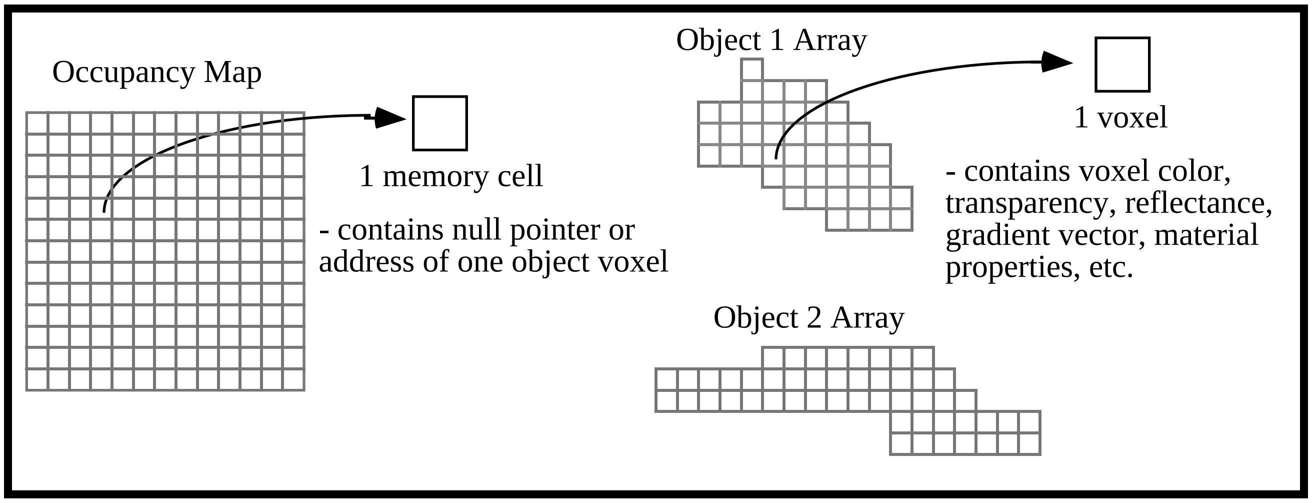
Occupancy Map
1 memory cell
- contains null pointer or address of one object voxel
Object 1 Array
1 voxel
- contains voxel color, transparency, reflectance, gradient vector, material properties, etc.
Object 2 Array

Figure 1. Data structures for a voxel-based object representation. The occupancy map consists of regularly spaced memory cells of integer size that contain either a null pointer or an address of one object voxel. The occupancy map is the size of the application's virtual world. The object arrays consist of data structures each containing the information for a single voxel. This voxel data structure can include many properties and attributes including voxel color, transparency, gradient vector, position vector, material properties, etc. Object arrays are not necessarily rectangular; object voxels are not necessarily evenly spaced or of equal sizes; and voxels are not necessarily ordered within the array. Since a voxel-based graphics system is very memory intensive, the purpose of this double data structure is to pack the large voxel structures of the objects as efficiently as possible while maintaining a map of the entire virtual space for modeling interaction between objects.

In the proposed system, where the virtual environment consists of a large virtual space containing several independent objects, this volumetric partitioning method can be especially efficient. As long as objects are stored as non-overlapping convex volumes (non-convex objects can be sub-divided to form convex sub-volumes), then intermediate images for each object can be rendered independently, and the final image can be generated by compositing these intermediate images.

There are several advantages to this object-based rendering method. First, volumes containing objects can be made as compact as possible so that storage is minimized and empty space in the virtual world is not rendered. Second, if the viewpoint does not change during a time interval, only objects that have moved during the interval need to be re-rendered. The new intermediate images are combined with unchanged intermediate images in the final compositing step. This can greatly reduce rendering times in dynamic systems. Third, the object-based data storage gives an intuitive subdivision of the data for distribution to multiple processors in parallel rendering algorithms. Finally, object data remain static within an object array even when the object moves about the virtual environment. Movement of an object relative to the virtual environment only requires an update to the coordinate mapping transformation between the object and the environment. Hence, objects are not resampled when they move, eliminating object degradation due to accumulated resampling errors.

Collision Detection

When modeling objects in a virtual environment, it is important to model interactions between objects in a physically realistic way. For example, in most cases, objects must be prevented from penetrating each other and they should react to collisions by exchanging momentum and energy and possibly by deforming on impact. The first step of modeling interactions between objects requires the detection of collisions between moving objects.

In surface-based graphical formats, objects are modeled as lists of polygons or other primitive surface elements. In this format, each element is essentially a set of mathematical equations or conditions. For example, a single three-sided polygon consists of an ordered list of three vertices. The polygon surface element is the intersection of three half planes defined by the polygon vertices. In order to detect a possible collision between two objects, each element in the first object must be checked for intersection with many elements in the second object. This amounts to solving for an intersection between two sets of mathematical equations and conditions for each possible combination of object elements. Reasonably complex objects may consist of many thousands of surface elements. Although algorithms have been introduced that reduce the number of element pairs that must be checked for intersection, collision detection between complex objects remains one of the major computational efforts in computer applications which perform physically realistic modeling or simulation [BARRAF94].

In contrast, collision detection for voxel-based objects is conceptually simple and does not require any mathematical analysis. As objects are moved in the virtual environment, object voxel addresses are simply shifted in the occupancy map array. Collisions are detected automatically when a voxel address from one object tries to write into an occupancy map cell that is already occupied by the voxel address of another object. This simple algorithm is illustrated in Figure 3.

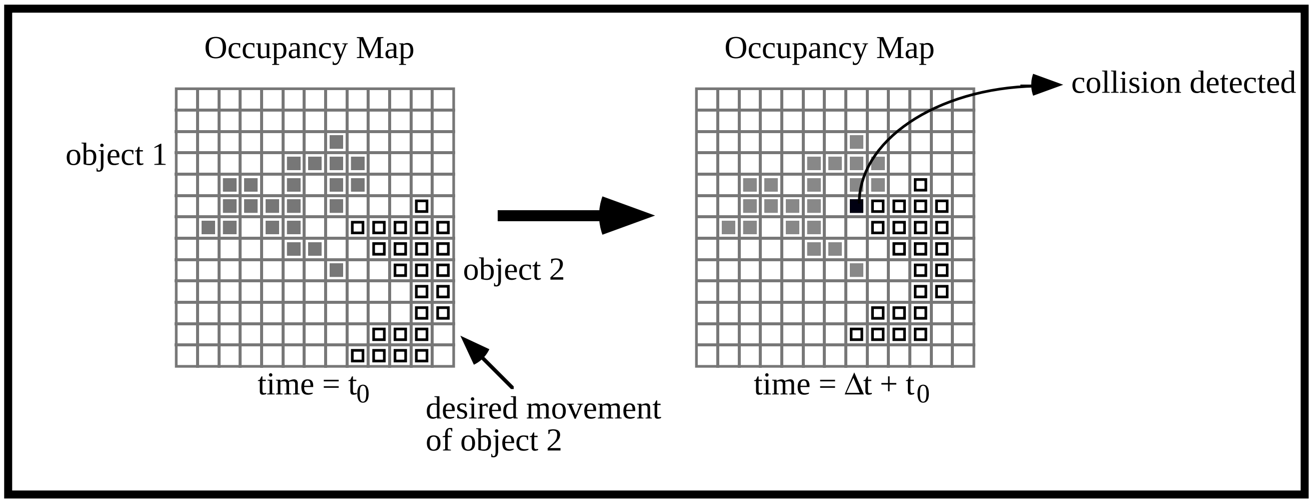

Figure 3. Collision detection algorithm. As object 2 moves along its desired trajectory, the addresses of the object voxels are shifted in the occupancy map. If an attempt is made to shift an object 2 address into an occupancy map cell that is already occupied by an address of object 1, then a collision is detected.

The data structures that have been proposed here suggest the following simple algorithm for detecting collisions and preventing objects from penetrating each other:

```
determine desired object position
while (stepsize to desired position > threshold) {
    for (all voxels in the moving object) {
        check occupancy map cells of desired positions to see if they are occupied
    }
    if (all occupancy map cells are free) {
        move the object into the desired position
        update the occupancy map
        exit
    }
    else if (any of the occupancy map cells is occupied) {
        avoid object interpenetration by reducing the step size
        update the desired position
    }
}
```

Collision Detection Prototype Systems

Two prototype systems have been built to test this algorithm for detecting collisions and avoiding object interpenetration. In a 2D prototype system, the occupancy map is updated as the positions of 2D objects are manipulated with the computer mouse. Applications that have been developed for this prototype include a virtual maze where objects with complex edges are moved between the voxel-based maze walls. Collisions with the maze walls are detected and the objects are restricted to the free space between the maze walls. In a second applications, 2D virtual puzzle pieces with arbitrary edges can be cut from a digitized color picture and assembled using a computer mouse.

In a 3D prototype system, voxel-based nut, washer, and bolt objects are interactively manipulated, visualized and assembled. The object positions and orientations are controlled interactively with a 6 degree-of-freedom Fastrack 3Ball input device by Polhemus. Computation and rendering was performed on an SGI Reality Engine although only minimal effort was made to optimize code or to take advantage of specialized hardware. Memory limits and the need for interactive rates limited the sizes of object volumes to less than about 680 kbytes. Volume rendering quality was compromised in order to attain interactive speeds (updates of 3 to 10 frames per second) but quality of the rendering was sufficient for good visualization of object position.

Using this 3D prototype system, objects can be moved, visualized and collided with walls and other objects at interactive rates. With some dexterity, it is possible to assemble the washer on the bolt, but poor positioning control and feedback with the current system makes it too difficult to thread the nut on the bolt. Planned improvements to the system include the addition of depth perception for better control of object placement along the z-axis, calculation of collision forces for simulating more physically realistic reactions to collisions, and force-feedback for better perception of collisions.

Haptic Exploration of Large Data Volumes

One direct application of the voxel-based algorithm for collision detection and avoidance of object interpenetration is a system to haptically explore or "feel" the 3D surface of a virtual object. Haptic, or tactile, exploration will enhance visualization methods for complex data and provide valuable feedback in the simulation of deformable objects. Using the proposed data structures along with a relatively small voxel-based model of the user's hand or fingertip, the computation required to detect and avoid collisions between the user's hand and the data is small even when the virtual object itself is very large and complex. In a voxel-based system for haptic exploration, the object data are stored in a large static data array. A small voxel-based object representing the user's hand or fingertip is shifted through the data array, tracking movement of the user's hand. When a voxel of the hand model encounters a voxel belonging to the object, a force feedback device is used to limit the hand movement to avoid penetration of the virtual object by the hand model. Using object-based volume rendering, pre-computed, high quality images of the volume data can be composited with dynamically computed images of the hand. As long as the hand model is small enough for fast rendering, visualization of the system will be fast enough for interactive use.

A prototype system, illustrated in Figure 4, is currently under development for haptic exploration of a high resolution 3D CT scan of a human hip. In the current design, a force feedback system [MASSIE94] is used to track the user's fingertip position in the virtual environment. The fingertip position is represented by a single voxel in the occupancy map. When the fingertip voxel encounters voxels belonging to the hip surface, the force feedback device prevents the user's fingertip from moving in a direction that would penetrate the virtual object, allowing the user to "feel" the hip surface. Stereo pairs of 3D volume rendered images of the hip data plus the fingertip are presented to the user via stereo goggles. Some control of the viewpoint is enabled in real time using pre-computed volume rendered images of the hip data.

Note: Figure 4 is hand-drawn and is available only in the Hardcopy version of this doccument.

Figure 4. A prototype system for haptic exploration of a virtual object using a force feedback device. Voxel-based algorithms for detecting collisions and avoiding object interpenetration are used as input to a force feedback device to limit the user's hand position as it explores the data volume. By restricting hand motion so that it can not penetrate the virtual model, the user is able to 'feel' the data.

Object Deformation

For modeling physically realistic interactions, we must be able to model both rigid and deformable objects. For example, in surgical simulation, we would like to be able to model deformation and tearing of skin during an incision, stretching or contraction of a muscle during limb movement, or deformation of an organ when it is probed by a surgeon's instrument. Some very interesting work has been done in modeling skin using finite element methods [TERZ90] [PIEP92]. However, finite element methods require solutions of large systems of differential equations and hence they are computationally expensive so that only small numbers of grid points may be used in interactive systems.

In a voxel-based deformable object model, each voxel is connected to its nearest neighbors. As forces are applied to the object, these connections are maintained while the object deforms. The connections act like non-linear springs which tend to pull the voxel mesh back towards an equilibrium state. Because the number of interconnections between voxels is large, finite element methods to determine final voxel positions at each time step are impractical. Instead, we allow the system to exist in a non-equilibrium state and adjust each voxel position independently of adjustments to neighbors. A closed feedback loop iteratively pulls each voxel towards an equilibrium position determined by the current position of neighboring voxels. This system will eventually settle in an equilibrium state after a finite disturbance if the feedback system is stable. Figure 5 illustrates a voxel that is pulled towards an equilibrium position equidistant between its 4 nearest neighbors.

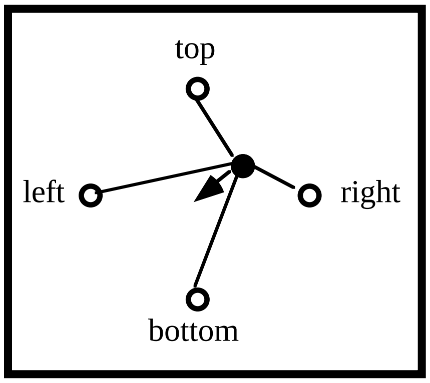

Figure 5. At each time step, the distances from the current voxel (dark circle) to its nearest neighbors (open circles) are examined. If these distances do not satisfy an equilibrium condition, the current voxel is moved closer to an equilibrium position. In this example, the current voxel is moved towards the point equidistant from its nearest neighbors.

The same data structures used for voxel-based rigid objects in a virtual environment are also used for deformable objects. However, in addition to color and other information, the object array elements also contain each voxel's exact position in the virtual environment. As the objects are deformed, these positions are updated and hence, although the object data remain static, the object deformation warps the data grid. The major advantage to this strategy is that the object data is not degraded by resampling errors when the object is stretched or compressed. However, rendering algorithms that can handle non-uniform grids are required. As in a non-deformable system, each occupancy map cell contains either a null pointer or the address of a voxel lying within the cell's area in the virtual environment. While a voxel can be physically located anywhere within a given cell, each cell may contain at most one voxel.

In a rigid system, the distance between neighbors cannot change. In an elastic system, the distance can change but will return to the zero state when equilibrium is achieved. In a deformable system, the system will still be in equilibrium when distances to neighboring voxels lie within some limits. Figure 6 illustrates the position that a voxel of a deformable object can assume relative to its left neighbor while remaining at equilibrium. As long as the voxel lies within the rectangular region defined by the maximum and minimum allowable horizontal separation and the maximum shear, $\text{horiz\_dy}_{\text{max}}$, in the vertical direction, then no position correction is required. Note that anisotropic materials can be modeled when different equalization regions are allowed for horizontal and vertical neighbors. For example, if the system of Figure 6 has the given equilibrium region for left and right neighbors and an equilibrium region for top and bottom neighbors where maximum and minimum separations along the vertical axis are equal, then the object would be able to compress and expand along the horizontal axis but not along the vertical axis.

Using this strategy, we can model materials such as skeletal muscle which can be more easily stretched and compressed along one axis than the other.

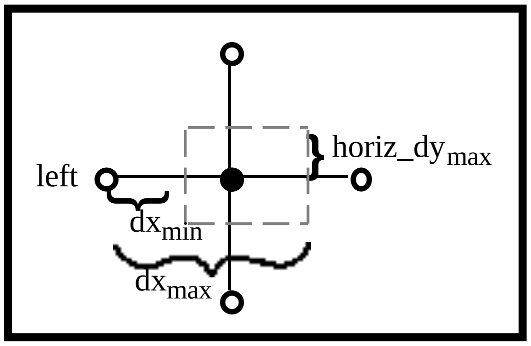

Figure 6. Equilibrium position of the current voxel (dark circle) relative to its left neighbor. As long as the current voxel lies within the outlined rectangle, then it is at equilibrium with respect to its left neighbor. If dx_{min} is equal to dx_{max} , then the object will not deform in the horizontal direction. The magnitude of horiz_dy$_{max}$ affects how much the object can shear.

In a deformable system, the motion of a selected voxel is limited by its surrounding voxels in the sense that a voxel cannot move into a cell that is already occupied. Hence, in order for a voxel to move from one position to another, all cells intersected by the line joining the two positions must be empty. When movement along a given path is inhibited by an occupied cell, the current voxel is moved as far as possible within its present cell and then the unfinished movement is transferred to the voxel in the occupied cell. The new voxel then attempts to continue movement along the desired path. This transfer of momentum continues until either the sum of the voxel movements is equal to the desired movement or a fixed voxel is encountered. As controlled voxels are moved, other voxel positions continue to be iteratively adjusted towards equilibrium positions. Figure 7 illustrates the movement of a selected voxel in a 1D system. Note that the deformable object is compressed when pushed and stretched when pulled.

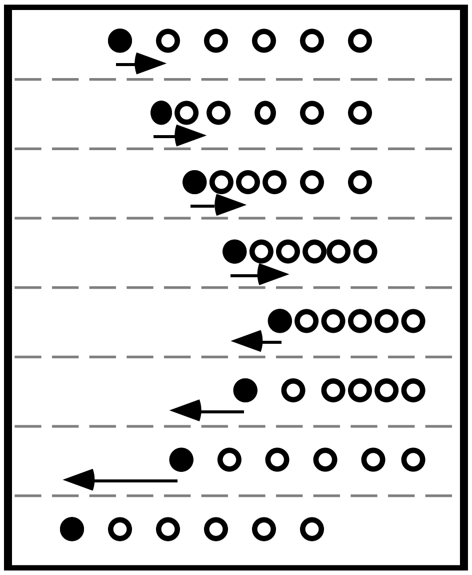

Figure 7. When the selected voxel (dark circle) is moved within a specified minimum separation from its neighbor, the minimum separation is maintained and both voxels continue to move in the desired direction. When the selected voxel is moved in the opposite direction, as in the bottom half of the figure, its neighboring voxel begins to move once a specified maximum separation occurs. When the minimum and maximum separations differ, the object will compress when pushed and stretch when pulled. The object's deformability is increased by increasing the difference between the maximum and minimum allowable separations.

Both 1D and 2D prototype systems have been built to test algorithms for moving and deforming voxel-based objects. While much work is needed to improve these systems and enable interactive control of reasonably sized objects, these prototypes have demonstrated that realistic simulation of complex motions can be accomplished with the data structures and strategies that have been presented here. Current efforts are directed towards designing a 2D system that will model interactions between several deformable 2D objects of reasonable size and complexity.

Summary

This paper has introduced data structures and algorithms for modeling interacting, deformable voxel-based objects in a virtual environment. Prototype systems have been developed that show the feasibility of these systems for interactive control and manipulation of complex objects. While these methods will initially be most attractive for models derived from volumetric data, they will also have some advantages in conventional graphics systems. There are many challenges for developing a large interactive virtual environment using voxel-based objects, including limits due to rendering speeds, data access speeds, and memory requirements. However, these challenges are analogous to those originally faced by conventional graphics. Further development of intelligent data structures, fast, parallel algorithms, and special purpose hardware will enable the exploration and realization of the full potential of volume graphics.

Bibliography

[ARRID90] S. Arridge, "Manipulation of Volume data for Surgical Simulation", in "3D Imaging in Medicine", eds. K. Hohne et al, Springer-Verlag, 1990.

[AVILA92] Avila, R., Sobierajski, L., Kaufman, A., "Towards a Comprehensive Volume Visualization System", Proceedings, IEEE Visualization '92, pp. 13-20, 1992.

[BARAF94] Baraff, D., "Rigid Body Simulation" in "An Introduction to Physically-Based Modeling", Course Notes 32, organizer: A. Witkin, Siggraph, 1994.

[CABRAL94] B. Cabral, Computer Graphics and Image Processing laboratory at SGI has developed real-time volume rendering of a 256x256x256 data volume using the SGI Reality Engine texture mapping hardware.

[CHEN89] L. Chen and M. Sontag, "Representation, Display, and Manipulation of 3D digital scenes and their Medical Applications", Computer Vision, Graphics, and Image Processing, 48, 1989, pp. 190-216.

[DIGIO94] A. DiGioia, T. Kanade, R. Taylor, eds."Proceedings of the First International Symposium on Medical Robotics and Computer Assisted Surgery", Shadyside Hospital, Pittsburgh, PA, 1994.

[GERB91] J. Gerber et al., "Simulating Femoral Repositioning with Three-dimensional CT", J. Computer Assisted Tomography, **15**, 1991, pp. 121-125.

[HOEHN90] Hoehn, K.H. et al, "3D-Visualization of Tomographic Volume Data Using the Generalized Voxel Model", The Visual Computer, **6**, February, 1990, pp. 28-37.

[HSU93] Hsu, W., "Segmented Ray Casting for data Parallel Volume Rendering", Proc. 1993 Parallel Rendering Symposium, ACM Press, 1993.

[KAUF93] Kaufman, A., Cohen, D., Yagel, R., "Volume Graphics", Computer, **27**, July 1993, pp. 51-64.

[KAUF87] Kaufman, A., "Efficient Algorithms for 3D Scan-Conversion of parametric Curves, Surfaces, and Volumes", Computer Graphics, **21**, July 1987, pp. 171-179.

[KAUF88] Kaufman, A. and Bakalash, R., "Memory and Processing Architecture for 3D Voxel-Based Imagery", IEEE Computer Graphics and Applications, pp. 10-23, 1988.

[KAUF90a] Kaufman, A.,ed., "Volume Visualization", IEEE CS Press, Los Alamitos, CA, 1990.
[KAUF90b] Kaufman, A., Yagel, R., Cohen, D., "Intermixing Surface and Volume Rendering", in *3D Imaging in Medicine: Algorithms, Systems, Applications*, K.H. Hoehne, H. Fuchs, and S.M. Pizer, eds., Springer-Verlag, Berlin, 1990, pp. 217-227.
[MA93] Ma, K-L, Painter, J., Hunsen, C., Krogh, M., "A Data-distributed Parallel Algorithm for Ray-traced Volume Rendering", Proc. 1993 Parallel Rendering Symposium, ACM Press, 1993.
[MASSIE94] T. Massie, K. Salisbury, "The PHANToM Haptic Interface: A Device for Probing Virtual Objects", Proc. ASME Symp on Haptic Interfaces for Virtual Environments and Teleoperator Systems", Chicago, Nov. 1994.
[NEUM93A] Neumann, U., "Volume Reconstruction and Parallel Rendering Algorithms: A Comparative Analysis", Ph.D. dissertation, Dept. Computer Science, U.N.C. Chapel Hill, 1993.
[NEUM93B] Neumann, U., "Parallel Volume Rendering Algorithm Performance on Mesh-Connected Multicomputers", Proc. 1993 Parallel Rendering Symposium, ACM Press, 1993.
[PIEP92] Pieper, S., Rosen, J., Zeltzer, D., "Interactive Graphics for Plastic Surgery: A Task-level Analysis and Implementation", ACM Proc. Interactive 3D Graphics, **3**, pp. 127-134, 1992.
[ROBB94] R. Robb, ed., "Visualization in Biomedical Computing 1994", SPIE **2359**, 1994.
[TERZ90] Terzopoulos, D., Waters, K., "Physically-based Facial Modelling, Analysis, and Animation", J. Visualization and Comp. Animation, **1**, pp. 73-80, 1990.
[YAGEL92] Yagel, R., "Realistic Display of Volumes", Image Capture, Formatting and Display, SPIE Vol. 1653, pp. 470-476.
[YASU90] Yasuda et al, "Computer System for Craniofacial Surgical Planning based on CT Images", IEEE Trans. on Med. Imaging, **9**, 1990, pp. 270-280.