Volume Graphics (2006)
T. Möller, R. Machiraju, T. Ertl, M. Chen (Editors)

Discontinuous Displacement Mapping for Volume Graphics

Carlos D. Correa[1] Deborah Silver[1] Min Chen[2†]

[1]Rutgers, The State University of New Jersey [2]University of Wales Swansea, UK

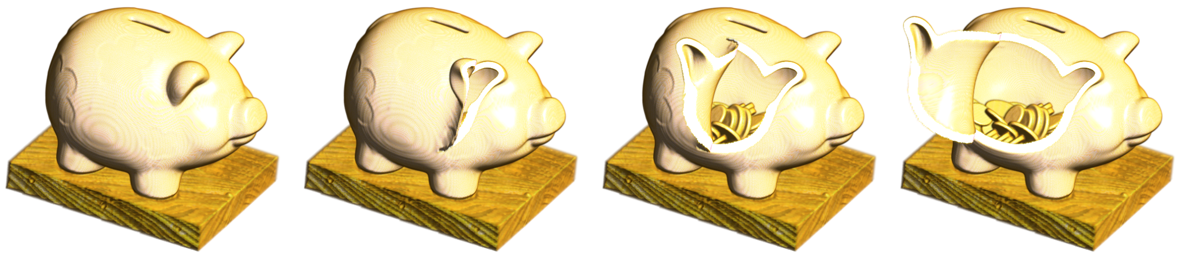

Figure 1: *Four temporal steps in which a dynamic discontinuous displacement map is applied to a piggy bank object interactively to simulate a cutting effect.*

Abstract
Displacement mapping is commonly used for adding surface details to an object. In this paper, we outline a generalized notion of displacement mapping, which allows for unconventional features such as unorthogonal and discontinuous displacement. By lifting the restriction on the geometric properties of the displacement, we can generate many different special effects including peeling, cutting and deforming an object. These types of operations are useful for volumetric objects, where the interior of objects is represented. To address the technical difficulties associated with this generalization, we employed inverse displacement maps in 3D vector space, and devised a collection of techniques, including sampling displaced objects through a proxy geometry, computing displaced surface normals, correcting lighting artifacts at breaking points in a discontinuous displacement map, and creating composite displacement maps from primitive maps on the fly. Through a number of examples of displacement maps, we demonstrate the generality, interactivity and usability of this approach on a set of volumetric objects.

Categories and Subject Descriptors (according to ACM CCS): I.3.1 [Computer Graphics]: Hardware Architecture-Graphics processors; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism.

1. Introduction

One of the key issues in volume graphics is to model interactive deformations including cuts and displacements. Existing methods for modeling and rendering deformations fall into two categories, physically-based and non-physically-based [CCI*05]. Physically-based deformation uses complex formulas to propagate forces over a mesh (e.g., finite element methods). Non-physically based techniques (see Section 2) use rendering mechanisms to achieve a similar effect. However, in the past, they have not been able to model cuts. Even for physically-based deformation, cuts are difficult to model because the mesh needs to be reconfigured. In this paper, we present a non-physically based technique to model cuts and deformations based on a generalized notion of displacement mapping. This technique can be directly deployed in many applications where real-time rendering is essential but realistic haptic feedback is not required. It can also be used in a physically-based deformation pipeline as a rendering engine after displacements are computed.

Displacement maps are commonly used to add visual details to a base surface by perturbing points on the surface for a small distance along the corresponding surface normals. For this reason, traditional displacement maps are generally (i) applied along the surface normal and (ii) assumed to be continuous. These conditions make it difficult to simulate large and complex deformations such as cuts and flipping-over. The first condition has been relaxed in some recent work.

† {cdcorrea,silver}@caip.rutgers.edu, M.Chen@swansea.ac.uk

Wang et al. [WWT*03] employed volumetric displacement functions in order to simulate non-orthogonal displacements on surface objects. Similar ideas are found in [WTL*04] and [PBFJ05]. These approaches, nonetheless, consider only displacements within small volumetric regions along the surface. The removal of the second condition is critical to rendering large cuts and breaks. Surface meshes do not contain adequate volumetric information, such as surface thickness and interior structure, to allow the creation of correct visual effects. Although this can be handled using a tetrahedral description of the interior, re-tessellation of such meshes is a time-consuming task, which limits the quality, smoothness and thickness of cuts and breaks.

In this paper, we introduce a generalized notion of a displacement map, which allows for unconventional features such as unorthogonal and discontinuous displacements. Figure 2 illustrates the difference between the traditional and generalized displacement mapping. We discuss the major technical difficulties associated with this generalization, and outline our solutions to these problems. In particular, we consider a GPU-based volumetric approach without involving any mesh structure and the intensive computation associated. By employing inverse displacement maps in 3D vector space, we are able to apply complex displacements to volumes. Our rendering approach involves the use of a proxy geometry for sampling of the inverse displacement map, which is then mapped into the original object space. Correct calculation of surface normals becomes a particular issue since conventional normal estimation would lead to noticeable lighting artifacts on surfaces displaced in varying directions, and at breaking points in a discontinuous displacement map. We show how to specify discontinuous maps in RGBα texture memory, and to render objects displaced under such maps on current GPU hardware. We adapted slicing-based rendering strategies, so that the deformed space is sampled in a view-oriented manner. We show how displacement maps can be combined to produce complex volumetric effects. Through these examples, we demonstrate the generality, interactivity and usability of this new approach, and its potential as a powerful technique for rendering many types of object interactions.

2. Related Work

Displacement Mapping. Displacement mapping was introduced by Cook [Coo84] as a type of texture mapping technique for modifying the geometry of a surface, resulting in correct shadows and silhouettes (in contrast to bump mapping [Bli78]). Typical approaches to the realization of displacement mapping include *explicit surface subdivision* (e.g., [CCC87]), *direct ray tracing* (e.g., [LP95,PH96]), and *image space warping* (e.g., [SP99]).

In surface subdivision [CCC87], geometric primitives are subdivided into micro-polygons, resulting in an explicit representation of the displaced surface. The method has been made available through commercial software such as RenderMan^{TM}and Maya^{TM}. Hardware solutions were also developed [GH99, DH00]. Discontinuities are introduced with costly re-meshing of the object. This approach cannot be extended easily for volume graphics, since no surface model is available.

In ray tracing, the conventional approach is to pre-compute an inverse displacement map, and perform the intersection calculation in the displaced space of a surface [LP95,PH96], similar to Barr's suggestion for rendering deformed objects [Bar86]. To alleviate the cost of ray-tracing an entire scene, recent approaches to displacement mapping propose to traverse rays through extruded triangles in the texture space [WWT*03, WTL*04, PBFJ05]. In image space warping, the visual effect of a displacement is achieved in the image space rather than the object space. The method was first introduced by Schaufler and Priglinger [SP99], focusing on warping the image of the base surface according to the projected displacement, and later extended by Oliveira *et al.* [OBM00]. The extension of ray tracing to the modeling of cuts is difficult, since it requires to determine a ray intersection with the new surface produced by the cut. Further, current approaches for displacement mapping based on extruded triangles cannot model large or discontinuous displacements. In our paper, we exploit the GPU capabilities of contemporary graphics boards to solve the limitations of the inverse displacement maps and image-space warping approaches when applied to discontinuous displacement on volumetric objects.

Non-physically based deformation. As an alternative to displacement maps, cuts and deformations have been explored as a modeling problem. The work presented here is not about physically-based deformation, or surgical simulation, for which there is a wide collection of literature (see [NMK*05] and [CCI*05]). While physically based deformations are useful for realistic haptic feedback, they require a huge computational cost which limits the complexity that can be achieved with desktop commodity hardware. Non-physically based deformation is a cost-effective alternative in many applications, such as illustrative, educational and entertainment software, where it is necessary to render complex deformations at interactive rates Existing non -physically-based methods in volume graphics include ray deflectors [KY97], spatial transfer functions [CSW*03], volume splitting [IDSC04], volume browsing [MTB03] and non-linear warping [BNG06]. Volume browsing uses point primitives to render cuts via forward transformations, which inherently allows for discontinuities. However, the use of points limits its quality in rendering, as undersampling occurs for large deformations. Ray deflectors and spatial transfer functions are ray-based approaches that use inverse mapping. They can be used to simulate cuts and other deformation effects such as magic lenses [WZMK05]. However, the issue of correct normal estimation along the cuts and breaks was not considered in the existing ray-based methods. It is also difficult for discrete ray casting to achieve

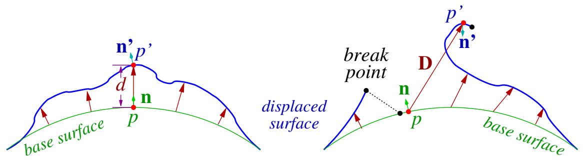

Figure 2: *A cross section illustration of the traditional displacement mapping (left) and the generalized displacement mapping allowing for unorthogonal and discontinuous displacement (right).*

the same level of performance as texture-based volume rendering without special data structures, which further complicates the simulation of cuts. In [BNG06], deformations are hard-coded in the GPU. An initial effort at inverse mapping is presented, but cuts cannot be achieved in real-time. Other deformation approaches, such as [WS01] and [RSSSG01] are based in the forward deformation of proxy geometry. It is difficult to extend these approaches for modeling cuts, since it would require a fine re-tessellation of the proxy geometry.

Both displacement mapping and texture-based volume rendering are desirable because of their suitability for GPU-based rendering. In this paper, we combine the best of both, remove the dimensional and geometric constraints normally associated with surface-based displacement mapping, and solve the technical challenges in the integration of generalized displacement mapping into a texture-based volume rendering pipeline.

3. Displacement Mapping

Displacement mapping is traditionally considered as a variation of 2D texture mapping, and it is used to alter the base surface geometrically. A volumetric displacement mapping can be considered as a variation of 3D texture mapping, where 3D displacements are used to perturb the volume, enabling the simulation of large deformations and cuts. Since texture mapping in 2D or 3D often involves textures defined in a higher dimensional parametric space, we consider a general notion of space Λ without explicitly distinguishing between geometry and texture spaces.

A Generalized Notation. Let Λ be a common reference coordinate system shared by an object position function P and an object displacement function $\overrightarrow{D}^{\triangleright}$. We consider the following generalized mapping from a point on the object $P(\lambda)$ to a new point $P'(\lambda)$

$$P'(\lambda) = P(\lambda) + \overrightarrow{D}^{\triangleright}(\lambda) \tag{1}$$

$\lambda \in \Lambda$ is a common reference. P defines a coordinate mapping from a common reference to a point in $\mathbb{E}^3$. $D^{\triangleright}$ is a vector function that can be specified procedurally or by using a discretized representation such as a texture. In Eq.(1), it is no longer a must that the surface normal determines the direction of displacement, and function $\overrightarrow{D}^{\triangleright}$ is no longer assumed to be continuous. As illustrated in Figure 2, in comparison with the traditional notion on the left, this generalized notion allows for unorthogonal and discontinuous displacement.

Λ can be a 2D parametric space as in the traditional notion, the 3D Euclidean space as in [KL96], or any reference system appropriate to an application. In this work, we use $\Lambda = \mathbb{E}^3$ as the common reference coordinate system. In the following discussions, we use mainly the inverse form of Eq. (1), that is,

$$P(\lambda) = P'(\lambda) + \overrightarrow{D}^{\triangleleft}(\lambda) \tag{2}$$

where $\overrightarrow{D}^{\triangleleft}$ is the inverse of $\overrightarrow{D}^{\triangleright}$. Since $\Lambda = \mathbb{E}^3$, we can make $P(\lambda) = \mathbf{p}$ and $P'(\lambda) = \mathbf{p}'$ where $\mathbf{p}, \mathbf{p}' \in \mathbb{E}^3$. By decomposing $\overrightarrow{D}^{\triangleleft}$ as $D(P'(\lambda)) = D(\mathbf{p}')$, Eq. (2) can thus be rewritten as

$$\mathbf{p} = \mathbf{p}' + D(\mathbf{p}') \tag{3}$$

For simplicity, we consider the backward displacement mapping function in the form of D in the following discussions. There is no constraint as to the displacement values of D. The displacement can be of any direction and magnitude.

4. Rendering

Our approach uses texture-based volume rendering with view-aligned slices. We use slicing as a rendering mechanism for discrete sampling of the displaced object space, rather than for storing a view-dependent representation of the displaced object. Since this space is unknown we define a bounding box object as a *proxy scene geometry*.

Let O be the function of an object and O' that of the displaced object. The proxy scene geometry in effect defines a bounded spatial domain of O'. If the proxy scene geometry contains a volume data set, we can use the texture-based volume rendering method which samples the proxy scene geometry with slices parallel to the view plane. Each pixel in a slice is then mapped back to the original object space where O is defined, via an inverse displacement map. Function O can be any object representation such that given a position $\mathbf{p}$ in the original object space, it returns appropriate luminance attributes at $\mathbf{p}$. $O(\mathbf{p})$ can easily be an implicit surface function, a level set surface, a distance field or a color volume texture. In our case, we use a scalar volume dataset, which we denote as O_j, and store it as a 3D texture in GPU memory. We also obtain a discretization of a displacement function, which we denote as D_k, from a procedural specification. D_k is stored as a texture of 8-bit or 16-bit point numbers, normalized in the interval $[-1, 1]$.

4.1. Displacement Setup

To create a displacement texture, D_k, of size $w \times h \times d$, we first specify $\overrightarrow{D}^{\triangleright}$ procedurally, and then sample its inverse transformation $D = \overrightarrow{D}^{\triangleleft}$ at discrete positions $\{x, y, z | x = 0, 1, \ldots, w; y = 0, 1, \ldots, h; z = 0, 1, \ldots, d\}$. For a discontinuous transformation, however, its inverse D that is analytically valid at all points does not exist, since some points in

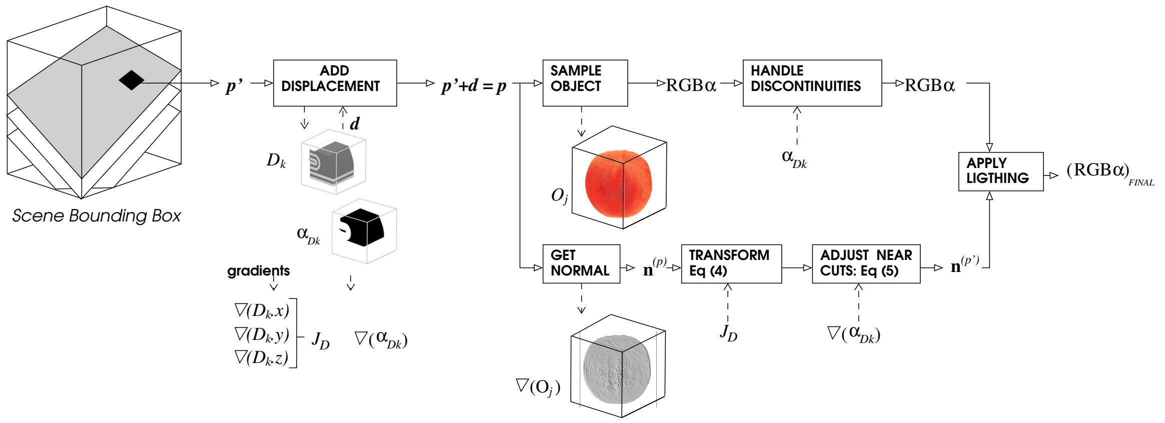

Figure 3: *System diagram for discontinuous displacement mapping. As we sample the bounding box of the scene, each fragment p' is displaced by a distance d obtained by sampling the displacement texture D_k. The resulting position $p = p' + d$ is used to sample the object texture O_j and gradient texture $\nabla(O_j)$, to obtain color and normal information. Color, normal and opacity (obtained from alpha mask α_{D_k}) are used to compute the final color of the fragment.*

the domain of D do not have a pre-image. For this reason, we compute a pseudo-inverse D_k, which is defined as the inverse D for those points in the co-domain of $\vec{D}^{\triangleright}$, and it maintains C^0 continuity.

In other words, D_k is valid for all points in the proxy scene geometry, including those where there should be a discontinuity. D_k is then normalized in the interval $[-1,1]$, then scaled and biased to fit in the range of valid values of GPU textures. In order to model the actual discontinuity, we define a mask α_{D_k}, such that $\alpha_{D_k}(\mathbf{p}')$ is 1 if $\mathbf{p}'$ has a pre-image in the co-domain of $\vec{D}^{\triangleright}$ and 0 otherwise, and discretize it in a 3D texture. D_k and α_{D_k} are illustrated in Figure 3, where the striped pattern in the displacement map is used to show the stretching that occurs at the discontinuity. The actual break is modeled as 0 values in the alpha texture. Having the displacement map as a continuous field and the alpha mask to model discontinuities, we have addressed the need for proper trilinear interpolation of displacements performed by the graphics hardware. If an arbitrary displacement value is used to represent an empty voxel (e.g., 1 or -1) due to a cut, voxels in the boundary of the cut would have displacement values interpolated between the displacement of the nearest voxel in the surface of the cut and this arbitrary value, which clearly results in visual artifacts.

4.2. Displaced Object Points

In order to determine the displaced volume, we slice the proxy scene geometry into view-oriented slices, as shown in Figure 3. The bounding box of O' can easily be found by combining the bounding boxes of the object(s) and their displacements. The slices are rendered in back-to-front order and finally composited using alpha blending. For each point $\mathbf{p}'$ on the slice, we must find the appropriate displacement, since there can be more than one displacement acting on the object (See Section 6 on composite maps). We use a fragment program to find the opacity and color values of a given pixel with texture coordinates $\mathbf{p}'$. This program computes $\mathbf{p} = \mathbf{p}' + D_k(\mathbf{p}')$, where $\mathbf{p}$ is the position in the original object O that corresponds to the texture coordinate $\mathbf{p}'$. It then samples the 3D texture of O at the position $\mathbf{p}$ and retrieve the color components. Finally, in order to handle discontinuities, it samples the alpha mask α_{D_k} at the position $\mathbf{p}'$ and modulate the pixel's color components with the mask. In order to avoid aliasing artifacts in the cut, the program considers a point as transparent if the alpha mask is less than 0.5, and the resulting pixel will not contribute to compositing. The process of the fragment shader is depicted in Figure 3.

4.3. Displaced Surface Normal

In order to properly shade the object, we need the normal information at each point. Since we store objects as volumes, normals can be obtained using finite differences or more sophisticated filters such as the Sobel operator. For interactive rendering, the gradient can be pre-computed and stored in a 3D texture. When processing fragments, a texture fetch of the gradient texture yields the normal of the voxel and its magnitude (This is shown as ∇O_j in Figure 3). There are three cases to be considered: the points that are not displaced, the displaced points, and those in the boundary of a discontinuity. For points not displaced, the pre-computed normals can be used directly. The other two cases are handled as follows.

4.3.1. Normals at Displaced Points

For a point undergoing displacement, we must obtain a new normal. Figure 4(b) shows the case where the pre-computed normal is used, which results in incorrect shading. In Figure 4(b), we peel the top of a piggy bank object (the light is above the piggy bank). We see the incorrect shading of the peeled surface. Since the surface was originally facing away from the light, it remains dark even though it is now facing the light after peeling. A more accurate shading of the surface can be seen in Figures 4(c) and 4(d) where the peeled surface is correctly shaded and toward the light.

Normals can be computed on-the-fly by sampling the neighboring voxels (after displacement) and applying finite differences. However, this method requires up to 6 (for central differences) additional displacement computations, which is computationally expensive. What is needed is a way to transform the undeformed normals on the fly without additional sampling. Barr [Bar84] describes a transformation of normals for a forward transformation.

Given F a forward mapping such that $\mathbf{p}' = F(\mathbf{p})$, the new normal $\vec{\mathbf{n}}^{(p')}$ is:

$$\vec{\mathbf{n}}^{(p')} = \det \mathsf{J_F} (\mathsf{J_F}^{-1})^\top \vec{\mathbf{n}}^{(p)} \tag{4}$$

where $\mathsf{J_F}$ is the 3×3 Jacobian of F. Our rendering approach requires and *inverse* mapping. Given $G = F^{-1}$ as that inverse mapping such that $\mathbf{p} = G(\mathbf{p}')$, Eq.(4) leads to:

$$\begin{aligned} \vec{\mathbf{n}}^{(p)} &= \det \mathsf{J_G} (\mathsf{J_G}^{-1})^\top \vec{\mathbf{n}}^{(p')} \text{ or:} \\ \frac{1}{\det \mathsf{J_G}} (\mathsf{J_G})^\top \vec{\mathbf{n}}^{(p)} &= \vec{\mathbf{n}}^{(p')} \end{aligned} \tag{5}$$

where $\mathsf{J_G}$ is the Jacobian of G. Since G is obtained via 3D displacement, as defined in Eq.(3), or more explicitly:

$$\mathbf{p} = G(x,y,z) = \begin{pmatrix} g_1(x,y,z) \\ g_2(x,y,z) \\ g_3(x,y,z) \end{pmatrix} = \begin{pmatrix} x + D_x(x,y,z) \\ y + D_y(x,y,z) \\ z + D_z(x,y,z) \end{pmatrix}$$

the Jacobian J_G is the derivative of G with respect to the spatial coordinates. Therefore:

$$\begin{aligned} \mathsf{J_G}(x,y,z) &= \begin{bmatrix} \frac{\partial g_1}{\partial x} & \frac{\partial g_1}{\partial y} & \frac{\partial g_1}{\partial z} \\ \frac{\partial g_2}{\partial x} & \frac{\partial g_2}{\partial y} & \frac{\partial g_2}{\partial z} \\ \frac{\partial g_3}{\partial x} & \frac{\partial g_3}{\partial y} & \frac{\partial g_3}{\partial z} \end{bmatrix} \\ &= \begin{bmatrix} 1 + \frac{\partial D_x}{\partial x} & \frac{\partial D_x}{\partial y} & \frac{\partial D_x}{\partial z} \\ \frac{\partial D_y}{\partial x} & 1 + \frac{\partial D_y}{\partial y} & \frac{\partial D_y}{\partial z} \\ \frac{\partial D_z}{\partial x} & \frac{\partial D_z}{\partial y} & 1 + \frac{\partial D_z}{\partial z} \end{bmatrix} \\ &= \mathsf{I} + \mathsf{J_D} \end{aligned}$$

where I is the identity matrix and $\mathsf{J_D}$ is the Jacobian of the displacement map. Then, Eq. (5) becomes

$$\vec{\mathbf{n}}^{(p')} = \frac{1}{\det(\mathsf{I} + \mathsf{J_D})} (\mathsf{I} + \mathsf{J_D})^\top \vec{\mathbf{n}}^{(p)}$$

Since normal vectors must be normalized, the $\frac{1}{\det(\mathsf{I}+\mathsf{J_D})}$ factor can be omitted. This leads to our normal transformation equation:

$$\vec{\mathbf{n}}^{(p')} = (\mathsf{I} + \mathsf{J}_D^{(p')})^\top \vec{\mathbf{n}}^{(p)} \tag{6}$$

This last step avoids the division for zero that might occur when the Jacobian is singular. However, the Jacobian is only singular at regions of breaks. In our approach, the displacement map is well defined and continuous for all points, and the breaks are handled by a different mechanism in the pipeline (see Figure 3). In practice, the Jacobian of the displacement map is approximated as the gradient of the displacement texture, i.e., $\mathsf{J}_D \approx \left(\nabla_{D_x}, \nabla_{D_y}, \nabla_{D_z}\right)^\top$, using finite differencing. For speed up, one can pre-compute the matrix $\mathsf{B} = (\mathsf{I} + \mathsf{J}_D^{(p')})^\top$ for each voxel in the displacement map, and store it as a 3D texture.

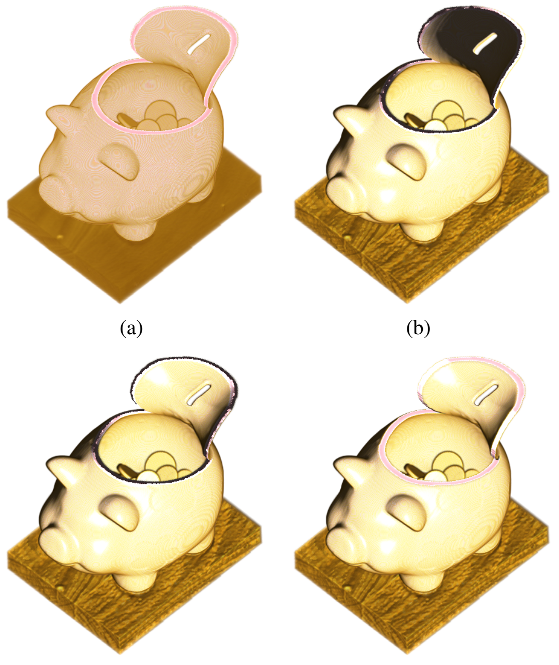

Figure 4: *Lighting computation for the piggy bank object. The light is above the piggy bank. (a) No lighting. (b) Using the pre-computed gradient results in incorrect lighting, notice how the underside of the cut surface is dark even though it is facing the light. (c) Correct lighting, but artifacts occur at discontinuities – rim of the cut area. (d) Correct lighting with proper handling of normals at discontinuities. Now the rim, which is facing the light, is lit properly.*

4.3.2. Normals at Discontinuities

Unfortunately, even with corrected normal transformation, we still see artifacts on the resulting objects at the boundaries of cuts. In this case, the normals of an object may change even when that part of the object does not undergo displacement. An example of this is depicted in Figure 4(c). Note that the rim of the piggy bank at the cut is dark. This is because those normals have not undergone transformations (like in the underside of the peeled surface) and are therefore incorrectly pointing away from the light source. This is especially noticeable on solid objects with a uniform interior. Even for the case of objects with a heterogeneous interior, as long as their normals are not directed orthogonal to the cut, the cut surface will be lit incorrectly.

To properly compute the normals at the discontinuities, we need a way to determine the new surface that has been created. This information is stored in the alpha map α_{D_k}. We can compute the gradient of the alpha mask $\nabla(\alpha_{D_k})$, and use this value only at the boundary of a cut. However, applying only this gradient in the boundary, may generate artifacts near the boundary. To solve this, we gradually correct the normal in the vicinity of the cut to the desired normal, via blending:

$$\overrightarrow{\mathbf{n}}^{(p')} = \omega(\mathsf{I}+\mathsf{J}_D)^\top \overrightarrow{\mathbf{n}}^{(p)} + (1-\omega)\nabla(\alpha_{D_k})^{(p')} \qquad (7)$$

where $\omega \in [0,1]$ is a blending factor. Figure 4(d) shows the result of applying this method for the piggy bank object. Note that the pixels at the rim of the cut are now properly shaded. This blending mechanism is similar to the solution proposed by Weiskopf et al. [WEE03] for volumetric cutaways. Although the alpha gradient can be computed on the fly using finite differencing, it can also be precomputed and stored in a 3D texture for speedup.

5. Construction of Displacement Maps

This section describes the process of creating a displacement map, using deformation of the bunny dataset as an example (Figure 8). This displacement uses a 3D Gaussian function to simulate a pull in the Z direction. The displacement can be constructed as:

$$D(x,y,z) = \begin{pmatrix} 0, & 0, & -ze^{\frac{(x-0.5)^2+(y-0.5)^2}{2\sigma^2}} \end{pmatrix}^\top \qquad (8)$$

for (x,y,z) in a unit cube, and σ chosen so that displacement becomes 0 at the XY boundaries of the unit cube. This displacement is discretized and stored in a 3D texture of size $64\times64\times64$. Note that the z value is used to modulate the amplitude of the Gaussian pull, and that the Z component of the displacement is negative, since D stores the *inverse* displacement. For instance, the point $(32,32,32)$ in the 3D texture, corresponding to the normalized point $(0.5,0.5,0.5)$ in the unit cube, contains the displacement vector $(0,0,-0.5)$.

When considering cuts, we follow a similar process. First, we compute the alpha channel of the cut procedurally, so that $\alpha_{D_k}(x,y,z)$ is 0 whenever there is a cut, and 1 elsewhere, and discretize it in a texture volume. In addition, we apply a smoothing operator over the alpha mask in order to obtain a smooth region around the boundaries needed for the blending of the normals, as described in Section 4.3.2. The result is stored in the alpha component of the displacement texture.

One challenge in the creation of displacements is the provision of user interfaces that would allow the user to define cuts and peels of arbitrary size and shape. In addition, user interface widgets are required to manipulate the different parameters of the displacement while rendering the volume. This is an important issue and it is ongoing research.

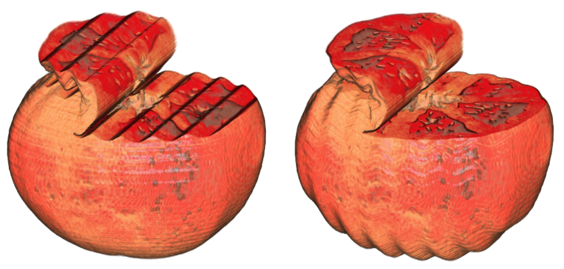
(a) $\mathbf{p} = \mathbf{p}' + D1(\mathbf{p}' + D2(\mathbf{p}'))$ (b) $\mathbf{p} = \mathbf{p}' + D2(\mathbf{p}' + D1(\mathbf{p}'))$

Figure 5: *Composition of two displacement maps $D1$ (wave) and $D2$ (peel) in different order.*

6. Composite Displacement Maps

In general, two transformations can be combined as follows:

$$\mathbf{p} = G_1(G_2(\mathbf{p}')) \qquad (9)$$

where $G_1(\mathbf{u}) = \mathbf{u} + D_1(\mathbf{u})$ and $G_2(\mathbf{v}) = \mathbf{v} + D_2(\mathbf{v})$ are displacement mappings.

For computing the normal, we must simply concatenate the Jacobians of the two mappings:

$$\overrightarrow{\mathbf{n}}^{(p')} = (\mathsf{B}_1 \times \mathsf{B}_2)\overrightarrow{\mathbf{n}}^{(p)} \qquad (10)$$

where $\mathsf{B}_1 = (\mathsf{I}+\mathsf{J}_{D_1})^\top$ and $\mathsf{B}_2 = (\mathsf{I}+\mathsf{J}_{D_2})^\top$ are the precomputed normal transformation matrices of the displacement mappings, as defined in Eq.(6). An example is shown in Figure 5, where two different displacements are combined in different order and applied to the tomato dataset. Since each displacement changes the frame of reference, composition is not commutative in general.

A powerful, yet rather simple, type of composition is through linear transformations. A linear transformation M can be defined as a 4×4 matrix, and can describe global rotations, translations or scalings of a coordinate frame. When applied to a displacement, it allows us to interactively place and scale the displacement map arbitrarily in the volumetric object, enabling effects as the one seen in Figure 1. From Eq.(9), we have:

$$\mathbf{p} = \mathsf{M} \times \left(\mathsf{M}^{-1}\mathbf{p}' + D(\mathsf{M}^{-1}\mathbf{p}')\right) = \mathbf{p}' + \mathsf{M} \times D(\mathsf{M}^{-1}\mathbf{p}')$$

The normal transformation is obtained by concatenating the inverse transpose of the Jacobians of M and M^{-1}.

Pre-computed combination of displacement maps results in another displacement map and does not require any changes in the GPU rendering process. It is a useful mechanism to create complex displacements from simple ones. On the other hand, on-the-fly combination enables the interactive creation and manipulation of independent displacement maps, though it requires a modified GPU implementation. Composite maps can be realized in the GPU program in a single pass by iterating the displacement procedure on the voxel positions for each displacement map, before sampling the original object. This approach, however, cannot be gen-

Dataset	Resolution	fps
Teddy Bear (Fig.7)	$256 \times 256 \times 224$	18.81
Bunny (Fig.8)	$256 \times 256 \times 256$	11.51
Piggy Bank (Fig.1)	$190 \times 190 \times 134$	7.52
Tomato (Fig.6)	$256 \times 256 \times 162$	6.24

Table 1: *Performance results for different volume datasets (sampling distance $d = 1.0$)*

eralized easily to many compositions. The research on a general multi-pass rendering process is being undertaken.

7. Results

We have implemented the displacement mapping approach within an interactive program which allows the user to rotate and scale the object and to move or change the displacement map. The displacement map can be changed interactively via linear transformations (see Section 6). Figure 1 shows a peeling of the size of a piggy bank dataset, revealing the coins in the inside. Figure 6 shows a slicing of a tomato dataset. Several time steps are rendered which show a progressive slicing as the user moves the displacement vertically. This displacement can be obtained from a simple slicing displacement and repeating it periodically. Figure 7 shows an opening of the teddy bear at the seam. Accurate lighting of the interior is required to avoid artifacts due to back facing normals. Figure 8 shows a Gaussian displacement to simulate a deformation of the Stanford bunny dataset. They can be placed and scaled anywhere in the scene, showing the flexibility of our approach to simulate other "traditional" deformations. Additional results are shown in the accompanying video, and at: http://www.caip.rutgers.edu/~cdcorrea/displacement/

8. Rendering Performance

Since we follow a slicing approach to render our scenes, rendering performance is mainly influenced by the fragment shader capabilities of the graphics board. The rendering speed is affected by a number of factors, including: sampling distance of the slices, resolution of the object, relative size of the displacement map, and viewport size. Our test configuration consists of a Pentium XEON 2.8 Ghz PC with 4096 MB RAM, equipped with a Quadro FX 4400 with 512MB of video memory. The performance results are shown in table 1, for a slicing distance of 1.0 and a viewport of size 512×512, for the displacements described in Section 7. Note that the performance is affected by the relative size of the displacement map. For instance, the teddy bear opening at the seam is smaller in size than the peeling of the piggy bank, and it is rendered at higher rates, even though the dataset is larger.

One aspect that affects the performance is the texture memory size and bandwidth. Volumetric displacement mapping requires the storage of the x, y, z components of the displacement. Although they can be stored in a single texture using 8 bits for each component, this precision is usually low for a smooth deformation, and results in visible jagged lines. In these cases, a 16-bit displacement can be used which would

Displacement	Resolution	Size in KB
Peel (e.g. Fig. 1)	$128 \times 128 \times 1$	320
Slicing (e.g. Fig. 6)	$128 \times 128 \times 1$	320
Extrusion (e.g. Fig. 8)	$32 \times 32 \times 32$	640
Seam opening (e.g. Fig. 7)	$64 \times 64 \times 64$	5120

Table 2: *Size in voxels of the displacement textures and texture memory requirement in total*

require at least two 3D textures. The first texture DISPXY stores the x and y components of the displacement as the luminance and alpha components, while the second texture DISPZA stores the z component and the α value (for discontinuities). This requirement does not pose a scalability problem in practice, since the use of general displacement maps allows the creation of complex cuts and deformations with relatively small 3D textures. Table 2 shows the size of the displacement textures used in this paper, and the total amount of texture memory required, which includes the storage of pre-computed Jacobians. Note that they are well within the limits of current GPU technology.

9. Conclusions and Future Work

In this paper, we have demonstrated how discontinuous displacement maps can simulate many different types of volume graphics effects such as fracturing, slicing, deforming, and cutting of graphical objects. We have employed inverse displacement maps in 3D vector space to solve for large and discontinuous displacements. We have also devised a collection of techniques, including computing surface normals changed due to unorthogonal displacement, correcting lighting artifacts at fractures, and creating composite maps from primitive maps on the fly. The displacement map can easily be represented as a 3D texture and the entire rendering process can be coded into a fragment program yielding interactive results. We have implemented several displacement maps and have shown their effect on a number of different models, demonstrating the generality, interactivity, and usability of this approach.

In this work, each displacement map is precomputed on the CPU from a procedural model. Such a model cannot capture the semantic information of the volume object to be manipulated, such as the thickness of a surface and a volume segment. Research effort is being made to incorporate semantic information through different feature-aligned displacement maps [CSC]. Further developments also include a displacement map editor for creating primitive and composite maps and a new version of user interface with novel interaction features. In addition to volume objects, the described technique can be applied to procedural volume models by substituting volume sampling with implicit function evaluation during object texture fetch.

Acknowledgment

Volumetric Datasets courtesy of the High Performance Computing and Communications, U.S. National Library of Medicine,University of Erlangen and Lawrence Berkeley Laboratory.

References

[Bar84] BARR A.: Global and local deformation of solid primitives. *Computer Graphics (Proc. SIGGRAPH 84) 18*, 3 (1984), 21–30.

[Bar86] BARR A.: Ray tracing deformed surfaces. *Computer Graphics (Proc. SIGGRAPH 86) 20*, 4 (1986), 287–296.

[Bli78] BLINN J. F.: Simulation of wrinkled surfaces. *Computer Graphics (Proc. SIGGRAPH 78) 12*, 3 (1978), 286–292.

[BNG06] BRUNET T., NOWAK K., GLEICHER M.: Integrating dynamic deformations into interactive volume visualization. In *Eurographics /IEEE VGTC Symposium on Visualization 2006* (2006), pp. 219–226.

[CCC87] COOK R. L., CARPENTER L., CATMULL E.: The Reyes image rendering architecture. *Computer Graphics (Proc. SIGGRAPH 87) 21*, 4 (1987), 95–102.

[CCI*05] CHEN M., CORREA C., ISLAM S., JONES M. W., SHEN P.-Y., SILVER D., WALTON S. J., WILLIS P. J.: Deforming and animating discretely sampled object representations. In *Eurographics State of the Art Report* (2005).

[Coo84] COOK R. L.: Shade trees. *Computer Graphics (Proc. SIGGRAPH 84) 18*, 3 (1984), 223–231.

[CSC] CORREA C., SILVER D., CHEN M.: Feature aligned volume manipulation for illustration and visualization. *To Appear*.

[CSW*03] CHEN M., SILVER D., WINTER A. S., SINGH V., CORNEA N.: Spatial transfer functions – a unified approach to specifying deformation in volume modeling and animation. In *Proc. Volume Graphics 2003* (2003), pp. 35–44.

[DH00] DOGGETT M., HIRCHE J.: Adaptive view dependent tessellation of displacement maps. In *Proc. EG/SIGGRAPH Workshop on Graphics Hardware* (Interlaken, Switzerland, 2000), pp. 59–66.

[GH99] GUMHOLD S., HÜTTNER T.: Multiresolution rendering with displacement mapping. In *Proc. EG/SIGGRAPH Workshop on Graphics Hardware* (1999), pp. 55–66.

[IDSC04] ISLAM S., DIPANKAR S., SILVER D., CHEN M.: Temporal and spatial splitting of scalar fields in volume graphics. In *Proc. IEEE VolVis2004* (Austin, Texas, October 2004), IEEE, pp. 87–94.

[KL96] KRISHNAMURTHY V., LEVOY M.: Fitting smooth surfaces to dense polygon meshes. In *Computer Graphics (Proc. SIGGRAPH 96)* (1996), pp. 313–324.

[KY97] KURZION Y., YAGEL R.: Interactive space deformation with hardware-assisted rendering. *IEEE Comput. Graph. Appl. 17*, 5 (1997), 66–77.

[LP95] LOGIE J. R., PATTERSON J. W.: Inverse displacement mapping in the general case. *Computer Graphics Forum 14*, 5 (1995), 261–273.

[MTB03] MCGUFFIN M. J., TANCAU L., BALAKRISHNAN R.: Using deformations for browsing volumetric data. In *Proc. IEEE Visualization 2003* (2003), pp. 401–408.

[NMK*05] NEALEN A., MULLER M., KEISER R., BOXERMAN E., M.CARLSON: Physically based deformable models in computer graphics. In *Eurographics STAR Report* (2005).

[OBM00] OLIVEIRA M. M., BISHOP G., MCALLISTER D.: Relief texture mapping. In *Computer Graphics (Proc. SIGGRAPH 2000)*. 2000, pp. 359–368.

[PBFJ05] PORUMBESCU S. D., BUDGE B., FENG L., JOY K. I.: Shell maps. *ACM Trans. Graph. 24*, 3 (2005), 626–633.

[PH96] PHARR M., HANRAHAN P.: Geometry caching for ray-tracing displacement maps. In *Proc. Eurographics Rendering Workshop* (1996), pp. 31–40.

[RSSSG01] REZK-SALAMA C., SCHEUERING M., SOZA G., GREINER G.: Fast volumetric deformation on general purpose hardware. In *Proc. SIGGRAPH/Eurographics Graphics Hardware Workshop 2001* (2001), pp. 17–24.

[SP99] SCHAUFLER G., PRIGLINGER M.: Efficient displacement mapping by image warping. In *Proc. Eurographics Rendering Workshop* (1999), pp. 175–186.

[WEE03] WEISKOPF D., ENGEL K., ERTL T.: Interactive clipping techniques for texture-based volume visualization and volume shading. *IEEE Trans. Vis. Comput. Graph. 9*, 3 (2003), 298–312.

[WS01] WESTERMANN R., SALAMA C.: Real-time volume deformations. *Computer Graphics Forum 20*, 3 (2001).

[WTL*04] WANG L., TONG X., LIN S., HU S., GUO B., SHUM H.-Y.: Generalized displacement maps. In *Proc. Eurographics Symposium on Rendering* (2004).

[WWT*03] WANG L., WANG X., TONG X., LIN S., HU S., GUO B., SHUM H.-Y.: View-dependent displacement mapping. *ACM Transactions on Graphics (Proc. SIGGRAPH 2003) 22*, 3 (2003), 334–339.

[WZMK05] WANG L., ZHAO Y., MUELLER K., KAUFMAN A. E.: The magic volume lens: An interactive focus+context technique for volume rendering. In *IEEE Visualization* (2005), p. 47.

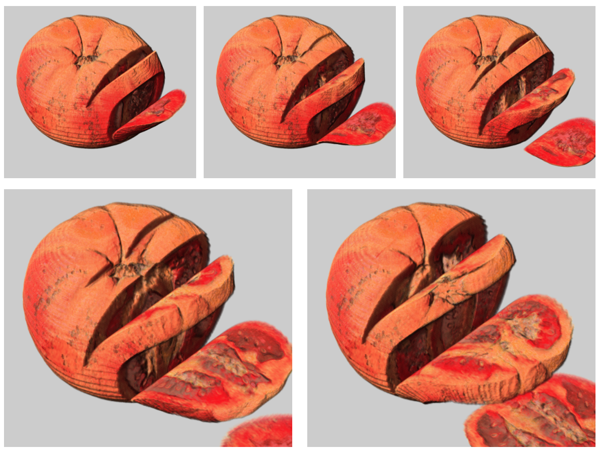

Figure 6: *Slicing of the tomato*

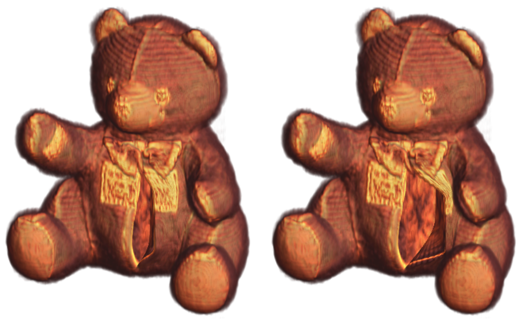

Figure 7: *Opening at the seam of the teddy bear dataset*

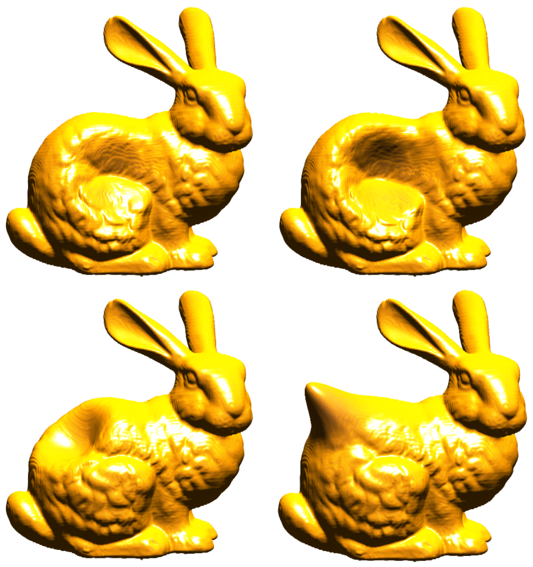

Figure 8: *Gaussian displacement map to simulate extrusion on the Stanford bunny CT dataset*