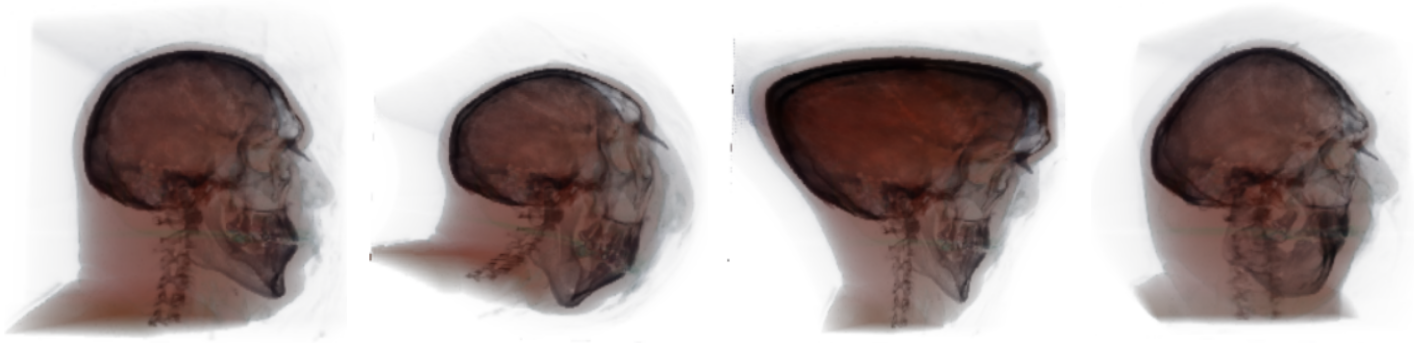
Deformation of Graphic Objects in GPU

Luciana Patricia das Virgens Silva
Departamento de Ciência da Computação
Universidade Federal da Bahia, UFBA
Salvador, Brasil
Email: lucianapvs@gmail.com

Vinícius Moreira Mello
Departamento de Matemática
Universidade Federal da Bahia, UFBA
Salvador, Brasil
Email: vinicius.mello@ufba.br



(a) Original (b) Bend (c) Taper (d) Twist

Fig. 1: Our method of inverse-ray deformation applied to the volume VisibleHuman. The first figure shows the original volume without deformation while the three following figures show the volume with the bend, taper and twist deformations respectively. The dataset used in this example is the conversion of the original file in PVM format to RAW format. The original data set used in this example can be found in [1].

***Abstract*—This paper proposes a method for deformation of graphic objects that incorporates vector field integration to the ray-casting algorithm in order to allow locally defined deformations.**[1]

***Keywords*-deformation; vector field; ray-casting**

I. Introduction

Geometric modeling can be seen as a subfield of Computer Graphics that encompasses the creation, representation and manipulation of graphic objects on the computer. This area virtually affects all areas of Computer Graphics and thus, in the formulation of solutions to problems related to modeling of graphic objects, it is necessary to take into consideration various factors such as the most suitable mathematical model for the problem, and the data structures and methods that would be better used to implement the solution on the computer.

A major goal of modeling an object or phenomenon is to allow a better understanding of the structure thereof. To achieve a better understanding sometimes it is not enough just to represent the object's form, it is also necessary to represent specific behaviours of the same when subject to manipulation.

This manipulations are called deformations and the graphic objects subjected to this process are called deformable objects. In the process of deformation, an object undergoes geometric and topological changes that will modify its shape. Such transformations are being increasingly studied and used in the area of Computer Graphics, especially in movies, animations and games with the aim of producing transitions in the appearance of characters.

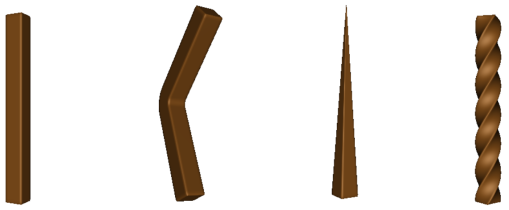


Fig. 2: Barr's traditional deformation methods applied to a mesh of triangles. The first figure shows the non deformed mesh of triangles and the following three the mesh with the bend, taper and twist deformations, respectively.

Contributions: This paper proposes a method for deformation of graphic objects that incorporates vector field integration to the ray-casting algorithm in order to allow locally defined deformations. The vector fields applied in this technique are based on the deformation methods proposed by Barr [2]. In addition, we developed a volume rendering tool to implement this deformation technique in GPU.

[1]This work relates to a M.Sc. dissertation

A. Related Work

Barr [2] introduced deformations through methods that perform operations to twist, bend or taper an object around a central axis (x, y or z), as we can see in the examples shown in Fig. 2. Thus, operations involving the movement of many control points could now be carried out with the change of a single parameter. Barr proposed two kinds of deformations: *global deformation*, where the warping function is explicitly known, and the *local deformation*, where a field of local transformations (jacobian matrices) must be integrated in order to obtain the global transformation. Note that he uses a matrix field, while we use the simpler concept of vector field. Since the pioneering work of Barr, many others methods for deformation were proposed. An extensive survey can be seen in [3].

II. Deformation of Graphic Objects

As previously stated deformation methods are widely used in various areas of Computer Graphics. There are several ways of specifying a deformation. The choice of method influences both the user interface and the deformation algorithm used.

A. Deformation Based On Vector Fields

The idea of deformation based on vector fields consists in construct a C^1 continuous vector field $\vec{v}$ and obtain the new positions of every vertex p of the shape by applying a path line integration of $\vec{v}$ starting from p. This approach does not generate self-intersections, what is a great advantage. Moreover, if the vector field is divergence-free, the deformation preserves volume. The paper [4] is a nice example of this strategy, nevertheless it applies the deformation to a triangle mesh, while we apply the deformation to volumetric objects.

B. Inverse-ray Deformation

Generally when applying deformations in graphic objects, new objects must be created from the original to represent the deformed one. This new object is then rendered and displayed to the user, as we see in Fig. 3.

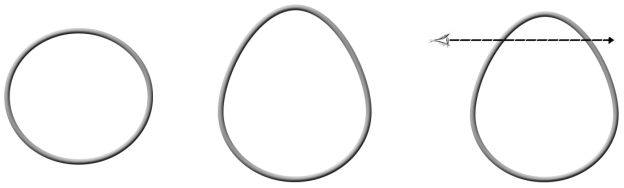


Fig. 3: Normal process of deformation of an object. First it is necessary to create the deformed object (middle) from the original (left) and only then run the ray casting (right).

But in the approach of inverse-ray deformation, the step of creating a new object can be eliminated since the deformation is applied to the rays released during the ray casting thereby making the process more efficient. Fig. 4 shows this process.

The original paper of Barr already proposed this technique. Hence, the present work can be seen as a implementation of Barr's deformation through vector fields and applied to volumetric objects using ray-casting algorithm combined with the inverse-ray technique.

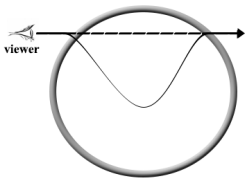


Fig. 4: Inverse-ray deformation technique. The deformation is applied directly on the ray in the opposite direction of the deformation movement to give a impression that the object is the one being deformed.

III. Infinitesimal Deformation of Graphic Objects

The theory of the infinitesimal deformation or small deformation theory can be regarded as a mathematical approach which describes the deformation of a solid body where the displacement of material particles are assumed to be infinitesimally small. Based on this theory we propose a deformation of graphical objects where the position of each point can be specified at each moment according to the vector field being used.

A. Vector Fields

We can define a vector field as a function that associates a single vector $\vec{v}(p)$ to each point p on plane or space. Thus, we have the following definition:

Definition 1. *Let A be a set in $\mathbb{R}^n$. A vector field on A is an application $\vec{v} : A \subset \mathbb{R}^n \to \mathbb{R}^n$. A vector field is C^1 if its coordinate functions are continuously differentiable.*

B. Local Deformation With Vector Fields

In this technique we combine ray casting with vector fields to create deformations in graphic objects. We define that the rays cast into the object will follow paths or streams delimited by predefined vector fields. These vector fields are adaptations of the methods proposed by Barr and that were shown previously. To construct this adaptations we use what we called basic vector fields. Those basic fields represent simple movements such as rotation, contraction and expansion.

All global transformations can be achieved by local transformations consisting of multiplying the basic vector fields by a function that depends on the height. Thus the function will change the speed of the fields according to the height (value of z). In Fig. 5 we can see that when we multiply the expansion vector field by the function $h(z) = z * k$, the speed of the vector field increases the extent to which the height increases and when the height decreases the speed does the same but it happens in such a fashion that the vector field starts to contract.

The twist vector field is achieved in a similar manner, as we see in Fig. 6, but in this case we use a rotation field. When we multiply the rotation field by the function $h(z) = z * k$,

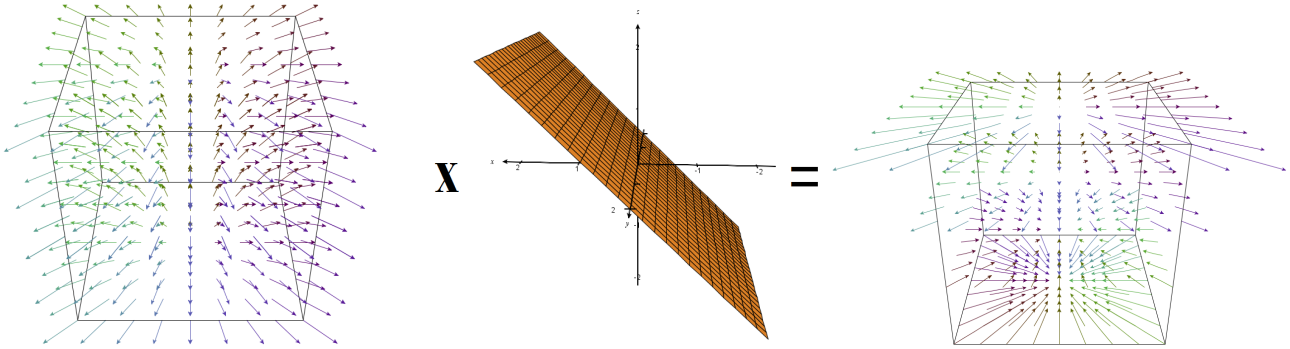


Fig. 5: By multiplying the expansion vector field $\vec{v}(x, y, z) = (x, y, 0)$, shown in the left image, by the function $h(z) = z * k$ (middle image) we get the taper vector field that we can see in the right image.

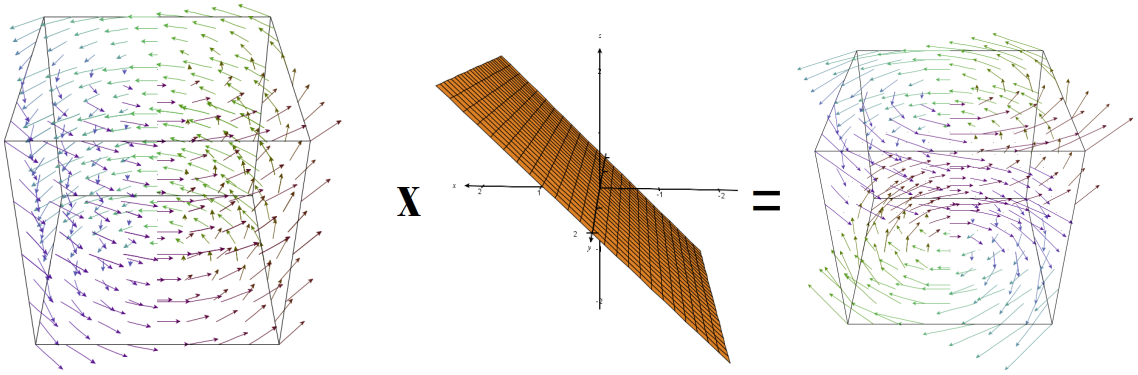


Fig. 6: Basic rotation vector field $\vec{v}(x, y, z) = (-y, x, 0)$ (left image) multiplied by the function $h(z) = z * k$ (middle image) results in the twist vector field shown in the right image.

the speed of the vector fields increases and decreases in such a way that the vector field spins in two different directions depending on the height.

To construct the bend field, we use a rotation vector field around the x-axis given by $\vec{v}(x, y, z) = (0, -z, y)$. The function used in this field can be either of the following:

$$h(z) = \arctan(10 * z)/(\pi/2)$$
or
$$h(z) = sign(z)$$

This function changes the speed of the vector field so that the field looks like it is leaning to its center thus given the impression of a bending, shown in Fig. 7.

With the vector fields already defined, we start describing the process of local deformation. The basic idea is to transform each point infinitesimally taking into consideration the following equation:

$$step(p) = p + dt \cdot \vec{v}(p),$$

In it we assume that dt is very small (infinitesimal) and the process will be iterated an n number of times so that $dt = t/n$. At the end of this process the point is taken to the position $I_t(p)$. We called the function I_t the flow associated with the field $\vec{v}$. To deform a mesh of triangle, for example, we simply apply the flow I_t to every vertex of the mesh.

It is interesting to note that the same process can be performed in the opposite direction to the field, in other words, in the direction of the field $-\vec{v}$, originating the flow I_{-t}. Now suppose we want to deform an implicit object $\mathcal{O} : f(q) = 0$. The deformed object is given by a set of points that satisfies a certain equation $\mathcal{O}' : g(p) = 0$. To know whether a point p in fact belongs to $\mathcal{O}'$, we have to determine if p is the image of some point q in $\mathcal{O}$ by the flow I_t, that is, $p = I_t(q)$. But since to reverse the flow we just need to walk in opposite direction of the field, $I_t^{-1} = I_{-t},$, we have that $q = I_{-t}(p)$, and

$$g(p) = f(q) = f(I_{-t}(p)).$$

For each step performed along the ray, a new loop will be added to know where the point will be taken by the deformation. This new loop is shown in the following algorithm.

Algorithm 1: Calculating $I_{-t}(p)$

```
while(t >= dt){
   p = p + dt * -v(p)
   t = t - dt
}
return p
```

To apply local illumination we need to calculate the gradient of the functions. One way to achieve this is using tricubic interpolation. Moreover, with the introduction of the deforma-

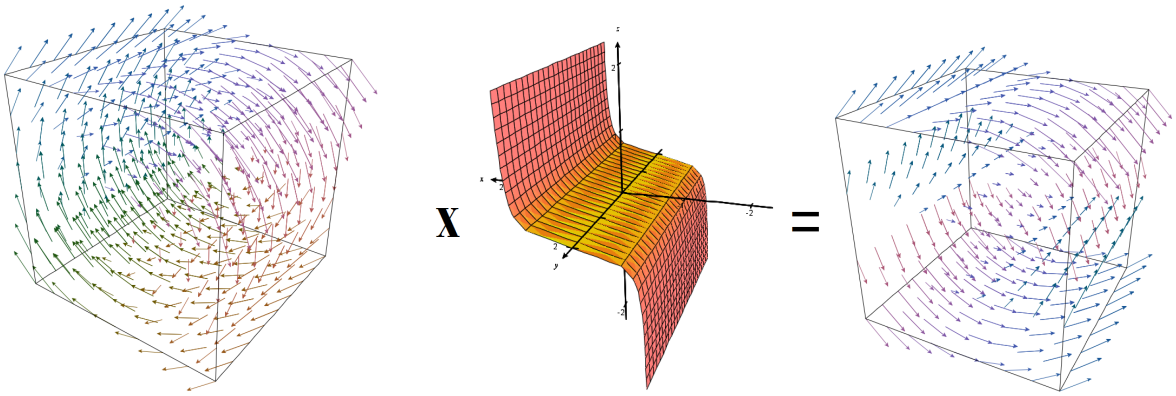


Fig. 7: A basic vector field representing a rotation around the x axis $\vec{v}(x, y, z) = (0, -z, y)$ (left image) multiplied by the function $h(z) = \arctan(10 * z)/(\pi/2)$ (middle image) creates the twist vector field (right image).

tion, it is also necessary to correct the gradient through an accumulation process described in Algorithm 2.

Algorithm 2: Accumulation process

```
D = id
while(t >= dt){
   D = (Id + dt * -Dv(p)) * D
   p = p + dt * -v(p)
   t = t - dt
}
return p, D
```

The math used to calculate the integration of the ray when walking in the field is given as follows: the point q is obtained by iterating the function $step$ n times, that is,

$$q = step^n(p),$$

with $n \cdot dt = t$. But

$$step^n(p) = step(step^{n-1}(p)).$$

Differentiating and using the chain rule, it follows that

$$Dstep^n(p) = Dstep(step^{n-1}(p)) \cdot Dstep^{n-1}(p).$$

On the other hand, as

$$step(p) = p + dt \cdot \vec{v}(p),$$

we have that

$$Dstep(p) = Id + dt \cdot D\vec{v}(p).$$

Therefore

$$Dstep^n(p) = (Id + dt \cdot D\vec{v}(step^{n-1}(p))) \cdot Dstep^{n-1}(p),$$

hence the Algorithm 2.

Finally, a curiosity. As $step(p)$ can be described as

$$step(p) = (Id(\cdot) + \frac{t}{n} \cdot \vec{v}(\cdot))(p),$$

it follows that

$$q = step^n(p) = (Id(\cdot) + \frac{t}{n} \cdot \vec{v}(\cdot))^n(p),$$

which reminds the definition of exponential thus, at least formally,

$$q \approx e^{t \cdot \vec{v}(\cdot)}(p).$$

IV. Volrend

Volrend is the tool created to implement this new technique and it is shown in Fig. 8.

A. Structure

This tool was based on Voreen [5], a open source volume rendering engine, and developed using the concept of dataflow networks. In summary, these networks are formed by modular units, also known as processors, which encapsulate the rendering algorithms and data processing. These units have input and output ports whereby the data will be exchanged with other units. Each door has a specific type. The connection between output and input ports will constitute the flow of data between the processors.

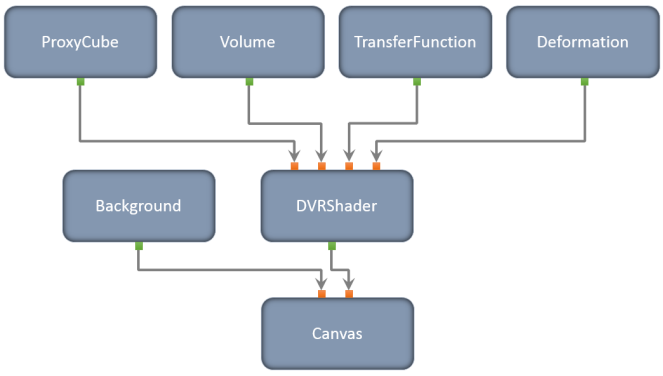


Fig. 9: Volrend's dataflow network. We chose this type of architecture because it provides more flexibility when creating and adding new components to the program.

Currently the dataflow network in this tool is composed of 7 processors (Fig. 9) and each one is responsable for a specific task. The processors were developed in Lua.

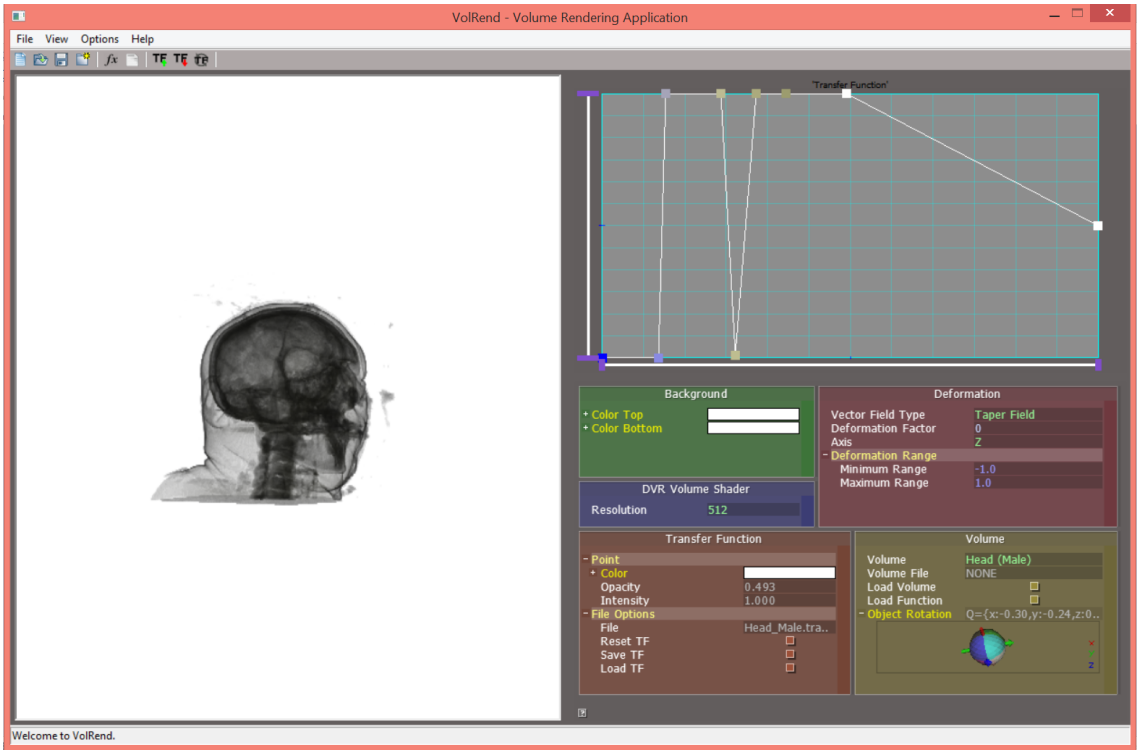


Fig. 8: The main display of Volrend. On the right side of the screen we see the canvas that shows the current rendering. On the upper left corner we see the configuration panel of the transfer function and under it are the menus used to configure the properties of the processors. Each menu is specific to a processor.

In Volrend's network, the first processor to run is the *ProxyCube*. This processor creates the textures that represent the entry and exit points of the ray. The *Volume* processor is responsible for loading the volume that will be rendered in the application. It displays a menu where the user can, for example, choose between the volumes already listed on the application or load a volume from an external source.

The *TransferFunction* processor takes the data entered by the user to create the transfer function. Next we have the *Deformation* processor that displays a menu to allow the user to set the deformation parameters. The *DVRShader* processor is one of the most important in the dataflow network. It triggers the execution of the shaders that are responsible for the process of volume rendering and deformation. It also shows a menu that allows the user to set the resolution of the image to be shown.

The last two processors, *Background* and *Canvas*, are responsible for the setting of the background colors and for show the results of the rendering on the screen, respectively.

B. Shaders

As our goal is to run the deformation technique in the GPU and enjoy the graphics capability of this device, we use the high-level programming language for graphics cards OpenGL Shading Language, also known as GLSL. With this language we can create shaders that are pieces of code that run on the GPU to manipulate the images before they are drawn on the screen. While the interface of the program was developed in Lua, all operations regarding the rendering and deformation of the volume were developed in GLSL.

V. Results and Discussion

To show the results of the application of this deformation technique we use datasets that contain regular volume data mainly coming from CT or MRI scanners. As mentioned before, it is possible to choose the resolution of the image displayed on Volrend. In Fig. 10 we can see the results with the three resolutions available, 128^2, 256^2 and 512^2. The greater the resolution the better the image.

Fig. 11: Teapot used in the examples. Original data set can be found in [6].

In the examples showed in Fig. 12 we apply the deformations to a CT scan of the SIGGRAPH 1989 teapot (Fig. 11).

It is also possible to define the region of the volume that it will be directly affected by the deformation, as shown in Fig.

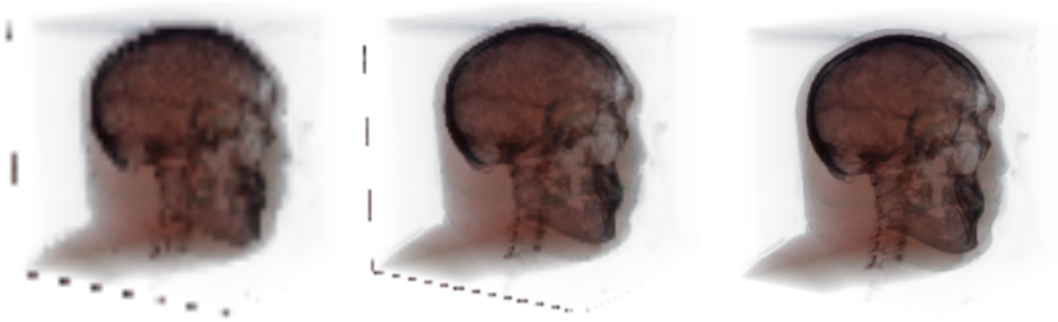


Fig. 10: Resolutions available on Volrend. Note that at lower resolutions (128^2), shown in the left image, there are multiple occurrences of subsampling problems and noise. In the middle image we see that these problems decrease as the resolution increases to 256^2 but we still notice some artifacts. With the 512^2 resolution, shown on the right, the image is way better.

(a) Bend Deformation

(b) Taper Deformation

(c) Twist Deformation

Fig. 12: Results achieved with the inverse-ray deformation method. We can see in (a), (b) and (c), different views of the application of our deformations methods to bend, taper and twist the volume.

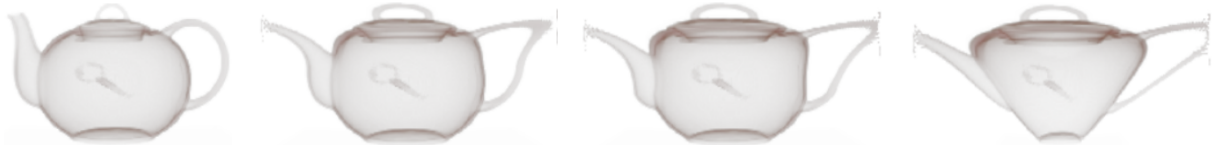


Fig. 13: In this example we show the taper deformation being applied in certain volume regions. The first figure is of the non deformed volume. The second one shows that only the upper half of the volume is being deformed. The third figure is of the volume with the deformation applied to 80% of its region. And the remaining figure shows the volume deformed in its totality.

13, by mutiplyng the deformation field by a field that is zero outside the region.

VI. Conclusion

In this paper, we showed a new technique to deform graphic objects using vector fields. We implement the deformation directly on the process of ray casting. The vector fields used in this method were adapted from the methods proposed by Barr. Here we apply the deformation method for implicit objects but this can also be used for parametric objects such as triangle meshes.

References

[1] "The volume library." [Online]. Available: http://www9.informatik.uni-erlangen.de/External/vollib/

[2] A. H. Barr, "Global and local deformations of solid primitives," *ACM Computer Graphics*, vol. 18, pp. 21–30, 1984.

[3] J. Gain and D. Bechmann, "A survey of spatial deformation from a user-centered perspective," *ACM Trans. Graph.*, vol. 27, no. 4, pp. 107:1–107:21, Nov. 2008. [Online]. Available: http://doi.acm.org/10.1145/1409625.1409629

[4] W. von Funck, H. Theisel, and H.-P. Seidel, "Vector field based shape deformations," *ACM Trans. Graph.*, vol. 25, no. 3, pp. 1118–1125, Jul. 2006. [Online]. Available: http://doi.acm.org/10.1145/1141911.1142002

[5] "Voreen: Volume rendering engine." [Online]. Available: http://www.uni-muenster.de/Voreen/

[6] "Volvis datasets library." [Online]. Available: http://www.volvis.org/