Isosurface Ray-casting for Autostereoscopic Displays

Balázs Domonkos, Attila Egri, Tibor Fóris, Tamás Juhász, and László Szirmay-Kalos
TU Budapest

ABSTRACT

In this paper the GPU implementation of a real-time isosurface volume-rendering system is described in detail, which aims at autostereoscopic displays. Since autostereoscopic displays provide images for many views, and thus require different camera settings in each pixel, and even in the three color channels of a pixel, naive rendering approaches would slow down the rendering process by a factor of the number of views of the display. To maintain interactive rendering, our approach is image centric, that is, we independently set the eye position for each pixel and implement iso-surface ray-casting in the pixel shader of the GPU. To handle the different camera settings for different color channels, geometric and color computation processes are decomposed into multiple rendering passes. This solution allows rendering rates that are independent of the number of main views of the autostereoscopic display, i.e. we cannot observe speed degradation when real 3D images are generated.

Keywords: Iso-surface ray-casting, autostereoscopic displays, GPU programming.

1 INTRODUCTION

3D autostereoscopic displays provide realistic depth perception for multiple viewers without any special aids like stereo glasses [9, 10]. To achieve the stereo effect, these displays emit spatially varying directional light, thus when a human observer looks at the screen with two eyes, the directions corresponding to the two eyes are slightly different, so are the perceived images. If these images are generated appropriately, the required stereo effect can be provided, allowing the viewers to move their head from side to side and see different aspects of the 3D scene. Current autostereoscopic displays equipped with parallax barriers or lenticular sheets placed on top of conventional screens offer a cheap and practical solution for 3D imaging [8].

Parallax barrier methods [5] use a fine vertical grating placed in front of the screen. The grating is made of an opaque material with fine transparent vertical slits at a regular spacing. Each transparent slit acts as a window to a vertical slice of the image placed behind it, and the exact slice depends on the position of the eye. *Lenticular displays* [1], on the other hand, separate images into different viewing directions using a sheet of long thin lenses.

Lenticular sheets contain a series of cylindrical lenses molded into a plastic substrate. The lens focuses on an image on the back side of the lenticular sheet. The lenticular image is designed so that each eye's line of sight is focused onto different strips.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

WSCG 2007

Plzen, Czech Republic.
Copyright UNION Agency – Science Press

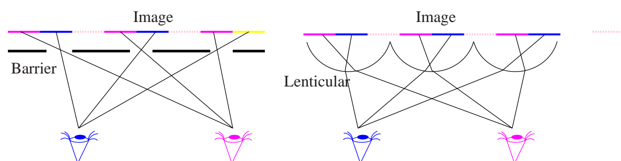

Figure 1: *Parallax barrier and lenticular displays*

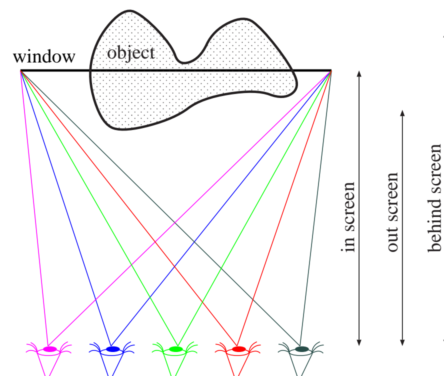

Figure 2: *Virtual camera model for autostereoscopic displays*

For these displays, the virtual scene should be rendered from several camera positions defined by the structure of the barriers or lenticules. Creating the several *subimages* requires the precise positioning of the camera and frustum. One common solution rotates the camera around a single point and symmetric camera frustums are used. This approach is quite popular since it is supported by conventional rendering systems. This is called the "toe-in" method [1]. However, it is not the correct method when a single flat physical display surface is used, and the viewing directions are not orthogonal to the surface. Applying the toe-in method on a flat surface causes not only horizontal but vertical parallax between the projections of the same spatial point. This distortion increases as one moves towards the cor-

ners. The correct approach is to offset the camera along a linear path and to use an offaxis frustum (figure 2).

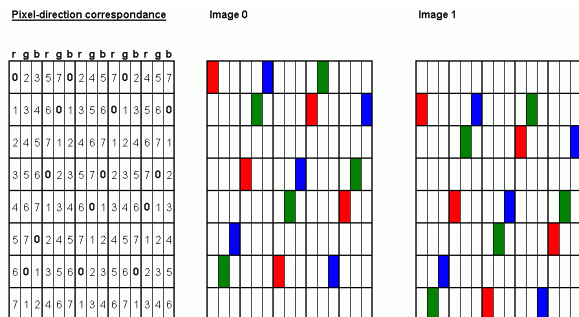

Figure 3: *Correspondence between the r,g,b channels of pixels and the 8 main directions, and the utilization of the first two subimages in the X3D-17 lenticular display. This pattern repeats itself on the screen and selects one direction from the possible 8, from where the particular pixels' r,g,b channels are visible. For example, the red, green, and blue points of the pixel at the left-top pixel are visible from directions 0, 2, and 3, respectively, and the same is true at every pixel whose horizontal and vertical distances from the left-top corner are multiples of 5 and 8, respectively.*

Since the lenses are usually not placed exactly vertically in order to reduce aliasing and abrupt changes in the image, the pixel-direction correspondence (figure 3) may change in every pixel row [12]. Furthermore, due to the fact that red, green, and blue emitting points have slightly different location in a single pixel, and lenses on these wavelengths have different index of refraction, the red, green, and blue channels of a single pixel correspond to different directions. These effects make the pixel-direction correspondence rather complex (figure 3), and prohibit the easy composition of low resolution subimages into a higher resolution display image. Therefore the subimages are usually computed on the same resolution as the final image, and the compositing procedure selects the red, green, and blue components for each target pixel from the subimages according to the pixel-direction correspondence (figure 3). This means that a larger part of original subimages are ignored during compositing (see the poor utilization of the first two subimages in figure x3d). If the bottleneck of rendering is fragment processing — which is usually the case in volume rendering and photorealistic image synthesis — this divides the speed by the number of main display directions (8–24 in current systems).

Using direct volume rendering, an isosurface can be implicitly extracted by resampling the volume data along the viewing rays at evenly located sample points [7, 11]. Rays are cast from the eye through the center of each pixel and the first samples, where the isosurface intersects the rays are determined.

This paper proposes an isosurface ray casting algorithm that does not increase the rendering time when the number of display directions gets higher. The speed degradation is avoided using an image centric rendering algorithm, such as ray-casting, and decomposing the rendering algorithm into geometric and spectral passes.

2 THE NEW ALGORITHM

Our approach decomposes the rendering process into two passes, one deals with geometric computations, while the other with spectral color data. This way we can solve the problem that the r,g,b channels of a pixel correspond to different camera positions without unnecessarily repeating the same geometric calculations. The geometric pass generates visible isosurface points and volume derivatives, such as the normal vector and curvature values, and derived values such as the cosine of the angle between the surface normal and the illumination direction. Formally, we assume that the surface reflection formula can be expressed in the following form

$$L(\lambda) = \sum_{i=1}^{n} a_i(\lambda) \cdot G_i \cdot I_i(\lambda)$$

where $a_i(\lambda)$ is the spectral property of the isosurface, $I_i(\lambda)$ is the spectral property of the light source, and G_i is the geometric property of both the illuminated surface and the light source. For example, if there is just a single directional light source and the diffuse + Phong-Blinn reflection model is used, then

$$G_1 = \cos\theta, \quad G_2 = \cos^m \delta$$

where θ is the angle between the gradient vector of the density field

$$\mathbf{g} = \nabla f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \right),$$

and the illumination direction, δ is the angle between the gradient vector of the density field and the halfway vector of the view and illumination directions, and m is the shininess of the material. On the other hand, concerning the spectral properties, $I_1(\lambda) = I_2(\lambda)$ are the intensity of the light source, isosurface spectral properties $a_1(\lambda)$ and $a_2(\lambda)$ are the diffuse and specular reflectances, respectively.

Note that if the cosines are negative, they should be replaced by zero.

2.1 Geometric pass

The geometric pass takes the definition of the virtual camera system and the 3D voxel array, and generates geometric properties $g_1, \ldots, g_n$ for each pixel of the screen. If n is not greater than 4, the result of the rendering pass is a floating point texture. Otherwise, we

should use the multiple render target option and store the result in more than one floating point textures.

Since the actual color channel of a pixel also affects the camera model, during geometric computations we assume that the wavelength corresponds to the red color channel.

Ray traversal

Comparing to conventional ray traversal, now we also have to find the eye position that corresponds to the processed pixel (figure 4).

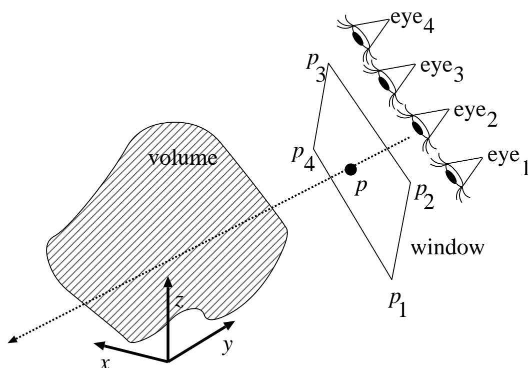

Figure 4: *When autostereoscopic camera model is used, we have to dynamically find the eye position for each pixel*

In this pass we render a single *full screen quadrilateral* (figure 5). To do that we set the model, view, and projection matrixes to be unit matrices, and pass a rectangle with vertices $(-1,-1,0)$, $(-1,1,0)$, $(1,1,0)$, $(1,-1,0)$. In the texcoord0 registers the 2D screen space pixel coordinates ($[0,0]$ to $[XM,YM]$) of the vertices are passed. On the other hand, texcoord1 registers encode vertices p_1, p_2, p_3, p_4 of the camera window in world space. The viewport resolution is set to the resolution of the display. This guarantees that the fragment shader is called once for every pixel. During fragment processing the interpolated texture coordinates will define pixel coordinates $[X,Y]$, and point p corresponding to this pixel in world space, respectively. Looking up the eye position of the red component of the particular pixel from the pixel-direction correspondence table (figure 3) using the interpolated pixel coordinates, we define the viewing ray from the eye and through world space point $\vec{p}$ corresponding to the current pixel.

This ray is intersected with the bounding box of the volumetric model to find entry point $\mathbf{p}_{entry}$ and exit point $\mathbf{p}_{exit}$ of the bounding box using the Cohen-Sutherland clipping algorithm. Then the ray is marched between the entry and exit points, evaluating sample points $\mathbf{p}_i$ as:

$$\mathbf{p}_i = \mathbf{p}_{entry} + (\mathbf{p}_{exit} - \mathbf{p}_{entry}) \cdot i/N, \qquad (1)$$

where N is the number of samples along the ray.

Density estimation

At each sample position $\mathbf{p}_i$ density $f(\mathbf{p}_i)$ of the volume is evaluated. Regarding the quality of isosurface rendering, the applied resampling technique to find $f(\mathbf{p}_i)$ is crucial.

As the volume data is a discrete representation, an appropriate reconstruction filter has to be applied to evaluate a density sample at an arbitrary sample position. Furthermore, the isosurface has to be shaded, therefore a surface normal, i.e. the first derivative, and sometimes curvatures, i.e. higher order derivatives, are calculated for each intersection point. The derivatives are obtained by resampling the volume by a derivative filter.

Generally the wider the support of the reconstruction filter is, the better its quality. On the other hand, by increasing the support of the filter kernel a convolution with it is getting more and more expensive computationally. In practical volume-rendering applications the most popular filter is the trilinear filter, since it represents a reasonable trade-off between quality and rendering speed. The most important drawback of trilinear interpolation, however, is that it produces discontinuous derivatives. Furthermore, some of the non-photorealistic volume-rendering techniques take also second derivatives into account, which can hardly be estimated by a linear filter. Therefore, to make our implementation generally usable with different rendering models, we apply a high-quality third-order (cubic) filtering technique proposed in [2].

The cubic reconstruction of a 1D signal can be formulated at an arbitrary position x as a weighted sum of the signal values at the nearest four sample positions:

$$f(x) \approx \tilde{f}(x) =$$

$$w_0(x) \cdot f_{i-1} + w_1(x) \cdot f_i + w_2(x) \cdot f_{i+1} + w_3(x) \cdot f_{i+2},$$

where $i = \lfloor x \rfloor$ is the integer part of x and $f_i = f(i)$ are the samples of the original signal. The filter weights $w_i(x)$ are periodic in the interval $x \in [0,1]$: $w_i(x) = w_i(\alpha)$, where $\alpha = x - \lfloor x \rfloor$ is the fractional part of x. Specifically,

$$\begin{aligned} w_0(\alpha) &= (-\alpha^3 + 3\alpha^2 - 3\alpha + 1)/6, \\ w_1(\alpha) &= (3\alpha^3 - 6\alpha^2 + 4)/6, \\ w_2(\alpha) &= (-3\alpha^3 + 3\alpha^2 + 3\alpha + 1)/6, \\ w_3(\alpha) &= \frac{1}{6}\alpha^3. \end{aligned}$$

The reconstructed function $\tilde{f}(x)$ can be evaluated as a linear combination of two linear texture fetches as follows:

$$\tilde{f}(x) = g_0(x) \cdot f_{x-h_0(x)} + g_1(x) \cdot f_{x+h_1(x)}, \qquad (2)$$

where

$$g_0(x) = w_0(x) + w_1(x),$$

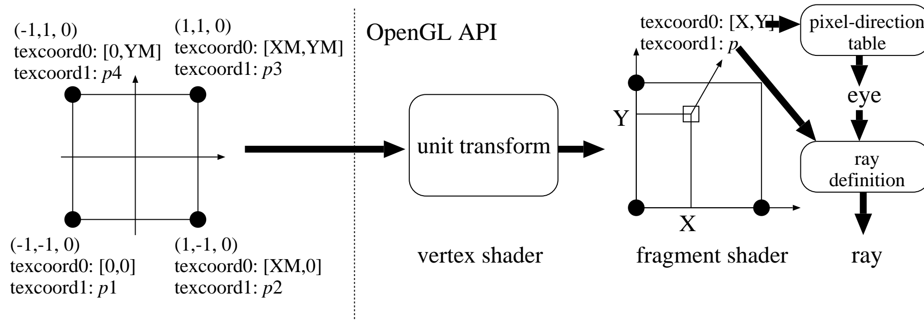

Figure 5: *Generation of the ray to be traced*

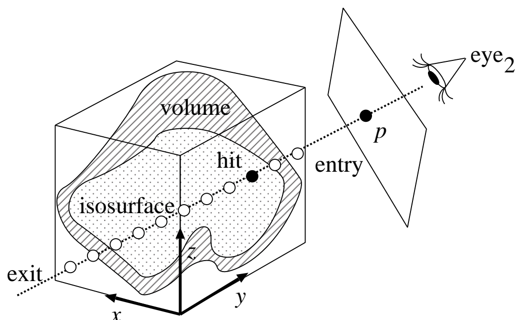

Figure 6: *Ray casting volume isosurfaces*

$$
\begin{aligned}
h_0(x) &= 1 - \frac{w_1(x)}{w_0(x)+w_1(x)} + x, \\
g_1(x) &= w_2(x)+w_3(x), \\
h_1(x) &= 1 - \frac{w_3(x)}{w_2(x)+w_3(x)} - x.
\end{aligned}
$$

Since functions $g_i(x)$ and $h_i(x)$ are also periodic, they can be stored in a lookup texture. In the 3D space the cubic reconstruction kernel is evaluated separately along the major axes and the resulting weights are simply multiplied (tensor product extension).

The samples of functions $h_0(x)$, $h_1(x)$, and $g_0(x)$ are stored in the x, y, and z components of an 1D texture. Function $g_1(x)$ does not need to be explicitly stored as $g_1(x) = 1 - g_0(x)$.

Isosurface intersection calculation

A ray-isosurface intersection is found between sample positions $\mathbf{p}_i$ and $\mathbf{p}_{i+1}$ if $f(\mathbf{p}_i) < s$ and $f(\mathbf{p}_{i+1}) \geq s$, where $f(\mathbf{p})$ denotes the density function and s is a threshold defining the isosurface. The ray traversal algorithm with isosurface intersection calculation is executed by the following fragment shader:

```
float3 raydir = pexit - pentry;
float  fv, fvprev;
float3 p, pprev;
// march the ray
bool found = false;
for(float t = 0; t <= 1.0f; t += dt) {
   if ( !found ) {
      float3 p = pentry + raydir * t;
      float fv = f(p);
      if (fv > s) { // intersection found
         found = true;
      } else {
         pprev = p;
         fvprev = fv;
      }
   }
}
```

Note that in this way we take the same number of samples along each ray between the entry and exit points of different distances, that is, the length of the steps is different for different rays. It seems to be a disadvantage, but current GPUs are much faster if they do not use the dynamic looping features. Step size `dt` is set to take at least one sample in each voxel along the ray.

A refined intersection point can be calculated by using the following root searching algorithm:

```
float3 pnew;
for(int n = 0; n < nIter; n++) {
   pnew = (p - pprev) * (s - fvprev) /
          (fv - fvprev) + pprev;
   fvnew = f(pnew);
   if (v < s) {
      pprev = pnew;
      fvprev = fv;
   } else {
      p = pnew;
      fv = fvnew;
   }
}
```

According to our experience 1-2 additional iteration steps provide sufficient accuracy.

This algorithm is mathematically equivalent to the numerical solution of equation $f(\mathbf{p}) = s$ starting a linear search then refining the solution with secant search.

Gradient estimation and calculation of geometric properties

After computing an accurate intersection point for each ray, the geometric factors are determined, which will be used by the next shading pass. The geometric properties depend on the gradient of the density volume, and possibly on second derivatives.

The gradient components are calculated by filtering the volume data with the partial derivatives of the 3D reconstruction kernel. For efficient derivative reconstruction the same fast filtering scheme can be used as for

 ISBN 978-80-86943-02-2

the function reconstruction with the following modifications. Now the weighting functions $w_i(x)$ sum up to zero instead of one, therefore $g_1(x) = -g_0(x)$.

The normalized gradients can be used for all shading models that require a surface normal, like the Phong-Blinn or tone shading. More sophisticated non-photorealistic or illustrative shading models, however, rely on second-order partial derivatives of the scalar field as well. The second-order derivatives yield the Hessian matrix [4], which characterizes the curvature in a given sample point:

$$\mathbf{H} = \nabla \mathbf{g} = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial x \partial z} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} & \frac{\partial^2 f}{\partial y \partial z} \\ \frac{\partial^2 f}{\partial z \partial x} & \frac{\partial^2 f}{\partial z \partial y} & \frac{\partial^2 f}{\partial z^2} \end{bmatrix}. \quad (3)$$

The first and second principal curvature magnitudes (κ_1, κ_2) of the isosurface can be estimated from the gradient $\mathbf{g}$ and the Hessian matrix $\mathbf{H}$ [6]. The principal curvature magnitudes amount to two eigenvalues of the shape operator $\mathbf{S}$ defined as the tangent space projection of the normalized Hessian:

$$\mathbf{S} = \mathbf{P}^T \cdot \frac{\mathbf{H}}{|\mathbf{g}|} \cdot \mathbf{P}, \quad (4)$$

where

$$\mathbf{P} = \mathbf{I} - \frac{\mathbf{g} \cdot \mathbf{g}^\mathbf{T}}{|\mathbf{g}|^2},$$

and $\mathbf{I}$ is the unit matrix. The eigenvalue corresponding to eigenvector $\mathbf{g}$ vanishes, and the other two eigenvectors are the principal curvature magnitudes. Because one eigenvector is known, it is possible to solve for the remaining two eigenvectors in the 2D tangent space without computing $\mathbf{S}$ explicitly [2]. This results in reduced number of operations and improved accuracy compared to the approach published in [6]. The transformation of the shape operator $\mathbf{S}$ to some orthogonal basis $(\mathbf{u}, \mathbf{v})$ of the tangent space is given by

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} = (\mathbf{u}, \mathbf{v})^T \cdot \frac{\mathbf{H}}{|\mathbf{g}|} \cdot (\mathbf{u}, \mathbf{v}). \quad (5)$$

Eigenvalues of $\mathbf{A}$ can be computed using the direct formulas for 2×2 matrices:

$$\kappa_{1,2} = \frac{1}{2}\left(\text{trace}(\mathbf{A}) \pm \sqrt{\text{trace}(\mathbf{A})^2 - 4\det(\mathbf{A})}\right), \quad (6)$$

This amounts to a moderate number of vector and matrix multiplications, and we have to solve a quadratic polynomial.

Having computed the gradient and the principal curvatures, the geometric properties are evaluated and stored in the target pixel. In our current implementation we store the geometric properties of the diffuse + Phong-Blinn reflection model ($G_1 = \cos\theta$, $G_2 = \cos^m \delta$). Using the principal curvatures the mean curvature ($G_3 = (\kappa_1 + \kappa_2)/2$) and the Gaussian curvature ($G_4 = \kappa_1 \kappa_2$) are obtained. Scalars (G_1, G_2, G_3, G_4) scalars can be conveniently packed into a single pixel.

If the ray happens not to intersect the isosurface, we put an invalid data item into the first channel, that is a $G_1 = -1$ is stored instead of the cosine of the angle between the surface normal and the illumination direction.

3 SHADING PASS

The shading pass is invoked after generating the image of the geometric properties, storing $\cos\theta$, $\cos^m \delta$, and the two curvature values in each pixel. Note that the geometric properties were obtained using the camera corresponding to the red channel of pixel. The geometric properties of the green and blue channels are stored in the neighboring fragments because of the shifts in the pixel-direction correspondence table (figure 3).

To complete shading, having changed the fragment shader program, a full screen quad has been rendered again. The fragment shader takes the geometric properties of the actual fragment, which provides information for the red channel, and based on the pixel-direction correspondence table, it also fetches the geometric properties for the green and blue channels from the fragments nearby using the table of figure 3. The fragment shader receives the spectral properties, such as the reflectances and light source intensities as uniform parameters, and the standard diffuse + Phong-Blinn reflection formula is evaluated. To add curvature information to the image, we take the two curvature values, consider them as a texture coordinate pair, and fetch a curvature color from a prepared lookup texture. This curvature color is added to the reflected illumination. Finally, the computed color is written into the frame buffer memory. Note that our shading pass is quite similar to *deferred shading* [3]. An important difference, however, is that we use also neighboring fragment data when a fragment is shaded since the data for blue and green channels are stored in other pixels.

4 RESULTS

The proposed algorithm has been implemented using OpenGL/Cg and run on an NV7800GT GPU. We used an X3D-17 autostereoscopic display that has a lenticular sheet in front of a 1280×1024 resolution 17 inch Fujitsu-Siemens TFT. The lenticular sheet is able to separate 8 main views. The zero parallax of the display is at 1.5 meter, that is the highest quality 3D images can be seen from this distance.

The visible human head used in our simulation has 512^3 resolution and a single voxel is stored in two bytes. The complete rendering, including isosurface localization, geometric property calculation, and shading

can be executed at 11 frames per second for a conventional 2D display screen, and the same rendering speed is attained when we used the proposed algorithm for the autostereoscopic display. It means that the additional overhead of reading the pixel-direction correspondence table is negligible. The beetle dataset has $256 \times 256 \times 128$ resolution and is rendered at 15 FPS.

The rendering speed is dominated by the geometric step, more precisely the computation of the ray isosurface intersection. In order to obtain higher rendering rates the intersection calculation should be speeded up by employing an empty space leaping scheme.

5 CONCLUSIONS

In this paper we presented an algorithm to interactively render volume isosurfaces onto a 3D autostereoscopic display. Since these displays trade off spatial resolution with directional dependence and thus 3D perception, the rendering time should not be higher than that of conventional 2D displays. However, previous approaches usually rendered the directional data on higher than necessary resolutions and composited the final image in the last pass. In this paper we showed that it is possible to eliminate unnecessary computations and obtain just as many pixel values that the underlying display surface has. To achieve this, we have to transfer all camera dependent computations to the pixel shader.

The proposed algorithm runs interactively for large volumetric models and does not slow down when 3D displays with many possible view directions are used.

ACKNOWLEDGEMENTS

This work has been supported by OTKA (T042735), GameTools FP6 (IST-2-004363) project, and by Hewlett-Packard and the National Office for Research and Technology (Hungary).

REFERENCES

[1] P. Bourke. Autostereoscopic lenticular images. Technical report, 1999. http://local.wasp.uwa.edu.au/~pbourke/stereographics.

[2] M. Hadwiger, C. Sigg, H. Scharsach, K. Bühler, and M. Gross. Real-time ray-casting and advanced shading of discrete isosurfaces. In *Proceedings of EUROGRAPHICS*, pages 303–312, 2005.

[3] S. Hargreaves and M. Harris. Deferred shading. Technical report, http://download.nvidia.com/developer/presentations/ 2004/6800_Leagues/6800_Leagues_Deferred_Shading.pdf, 2004.

[4] J. Hladuvka. *Derivatives and Eigensystems for Volume-Data Analysis and Visualization*. PhD thesis, Institute of Computer Graphics, Vienna University of Technology, Vienna, Austria, 2002.

[5] S.H. Kaplan. Theory of parallax barriers. *Journal of SMPTE*, 59(7):11–21, 1952.

[6] G. Kindlmann, R. Whitaker, T. Tasdizen, and T. Möller. Curvature-based transfer functions for direct volume rendering: Methods and applications. In *Proceedings of IEEE Visualization*, pages 513–520, 2003.

[7] M. Levoy. Efficient ray tracing of volume data. *ATG*, 9(3):245–261, 1990.

[8] W. Matusik and H. Pfister. 3D TV: A scalable system for real-time acquistion, transmission and autostereoscopic display of dynamic scenes. *ACM Transactions on Graphics (TOG) SIGGRAPH*, 23(3):814–824, 1993.

[9] T. Okoshi. *Three Dimensional Imaging Techniques*. 1976.

[10] S. Pastoor and M. Wopking. *3-D displays: A review of current technologies*. Displays 17, 1997.

[11] T. Theußl, O. Mattausch, T. Möller, and M. E. Gröller. Reconstruction schemes for high quality raycasting of the body-centered cubic grid. *TR-186-2-02-11, Institute of Computer Graphics and Algorithms, Vienna University of Technology*, 2002.

[12] x3d Technologies. *17" 3D-Display A3*. Opticality, 2005.

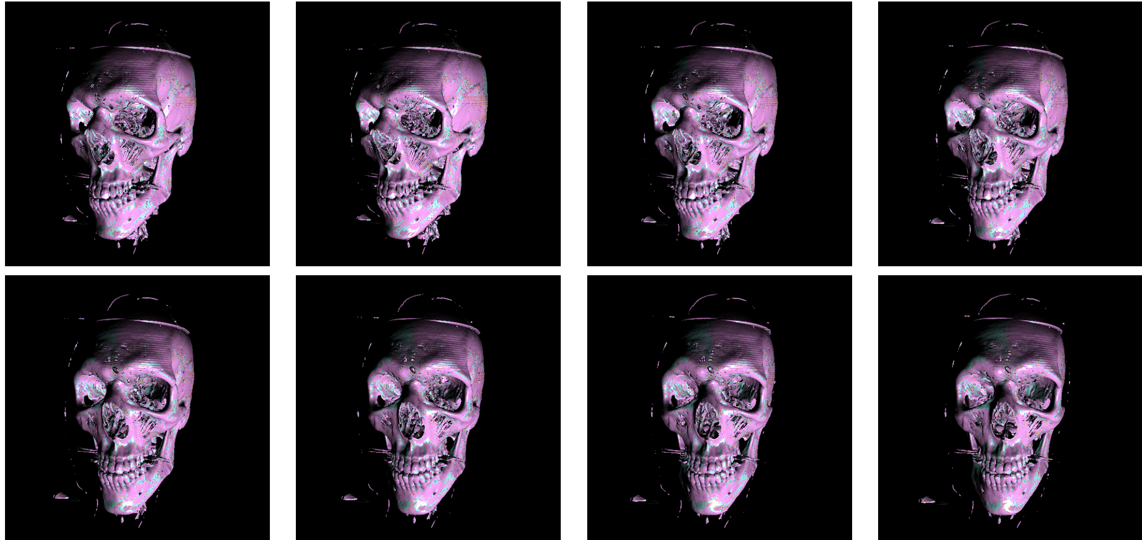

Figure 7: The human head dataset rendered from the main view directions setting the isovalue to visualize the bone

Figure 8: The beetle dataset rendered from the main view directions setting the isovalue to visualize the bone

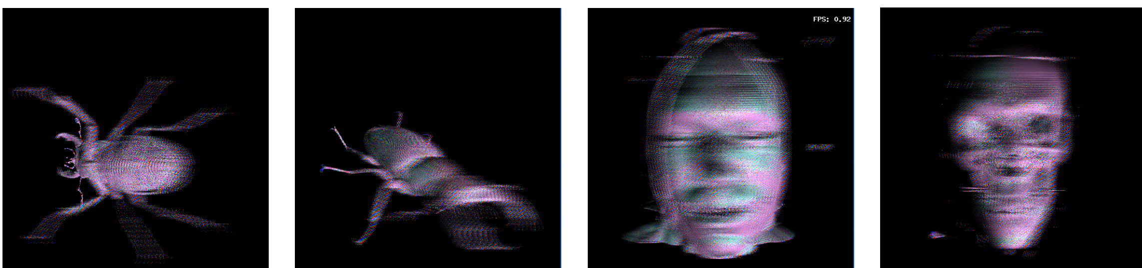

Figure 9: 3D autostereoscopic images when rendered on a normal 2D screen

 ISBN 978-80-86943-02-2