Hardware-accelerated Autostereogram Rendering for Interactive 3D Visualization

Christoph Petz and Bastian Goldlücke and Marcus Magnor

Max-Planck-Institut für Informatik
Graphics - Optics - Vision
Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany
Email: petz@mpii.de

ABSTRACT

Single Image Random Dot Stereograms (SIRDS) are an attractive way of depicting three-dimensional objects using conventional display technology. Once trained in decoupling the eyes' convergence and focusing, autostereograms of this kind are able to convey the three-dimensional impression of a scene. We present in this work an algorithm that generates SIRDS at interactive frame rates on a conventional PC. The presented system allows rotating a 3D geometry model and observing the object from arbitrary positions in real-time. Subjective tests show that the perception of a moving or rotating 3D scene presents no problem: The gaze remains focused onto the object. In contrast to conventional SIRDS algorithms, we render multiple pixels in a single step using a texture-based approach, exploiting the parallel-processing architecture of modern graphics hardware. A vertex program determines the parallax for each vertex of the geometry model, and the graphics hardware's texture unit is used to render the dot pattern. No data has to be transferred between main memory and the graphics card for generating the autostereograms, leaving CPU capacity available for other tasks. Frame rates of 25 fps are attained at a resolution of 1024x512 pixels on a standard PC using a consumer-grade nVidia GeForce4 graphics card, demonstrating the real-time capability of the system.

Keywords: SIRDS, Autostereogram, 3D Visualization, Graphics Hardware, Interactive Display

1. INTRODUCTION

For a classical stereo pair, a scene is photographed from two cameras that are translated relative to each other by a small amount, emulating distance of the human eyes. The stereogram implicitly contains scene depth information: The parallax between two pixels corresponding to the same 3D-point P becomes smaller the further P is away from the image plane. By finding these correspondences, the Human Visual System (HVS) unconsciously provides depth information when each eye is confronted with one of the images of such pair. In 1962, Julesz[1, 2] demonstrated that the HVS can perform this complex task purely from stereopsis, i.e. without relying on any other clues such as perspective or contours using *Random Dot Stereograms (RDS)*.

Tyler and Clarke[3] found a way to integrate the two images of an RDS into a single image, rejoicing in the name *Single Image Random Dot Stereogram (SIRDS)* or *autostereogram*. They devised an intricate scheme to color the pixels, which we describe in more detail in the next section. To decompose a SIRDS back into the left- and right-eye images, the eyes have to be trained to decouple *convergence*, which is the point where the lines of sight of both eyes cross, from *focus*, the depth to which the lenses adjust to. Once this is achieved, it is possible to display sequences of autostereograms without having to adjust the visual system anew for each frame. This allows for the creation of autostereogram animations, which are the primary objective of this paper.

Computational models of the visual processes that are involved in interpreting random dot stereograms were first discussed by Marr and Poggio[4] from the viewpoint of the psychology of stereopsis. Nowadays, algorithms for rendering single autostereogram images of a static scene, mostly based on the work by Thimbleby et al,[5] are widely available.[6] The pleasant images they produce inspired many artists to create innovative pictures, which found their way to an enthusiastic public in the form of posters, postcards, calendars and even books selling hundred thousands of copies.[7, 8] However, traditional methods for the generation of autostereograms are too slow to create images of animated scenes at interactive frame rates. We present a hardware-accelerated algorithm

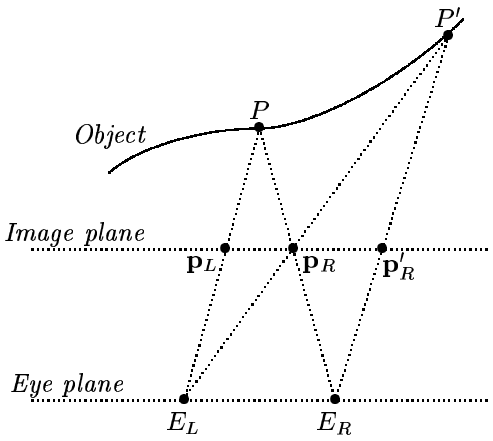

Figure 1. Interacting pixels $\mathbf{p}_L$, $\mathbf{p}_R$ and $\mathbf{p}'_R$ in the image.

which exploits the customizable vertex transformations and texturing capabilities of modern consumer-market graphics hardware to render SIRDSs of complex scenes in real-time. This enables using SIRDS animations as an innovative visualization tool to convey 3D structure of complicated geometry models on any desktop PC, without the need for expensive specialized hardware.

We briefly introduce the general theory of autostereograms in Sect. 2. Our novel rendering algorithm is described in Sect. 3, followed by a proposed hardware-accelerated implementation in Sect. 4. Experimental results are presented in Sect. 5. We conclude with some ideas for applications and future work in Sect. 6.

2. GEOMETRY OF AUTOSTEREOGRAMS

The geometry of the scene and the viewing system imposes restrictions on the colors of pixels in the autostereogram. For any given scene point $P \in \mathbb{R}^3$ which is visible from both eyes, we have the projections $\mathbf{p}_L$ and $\mathbf{p}_R$ of P into the image of the left or the right eye, respectively, Fig. 1. Since $\mathbf{p}_L$ and $\mathbf{p}_R$ are the projections of the same scene point P, they must have the same color. We want to construct a *single image* random dot stereogram, so we have to iterate this process: $\mathbf{p}_R$ is also visible from the left eye, so it can be viewed as the projection $\mathbf{p}'_L$ of a different scene point P', which in turn has again a projection $\mathbf{p}'_R$ when seen from the right eye. The same conclusion as before now yields three points $\mathbf{p}_L$, $\mathbf{p}_R = \mathbf{p}'_L$ and $\mathbf{p}'_R$ in the image plane which have to be assigned the same color.

When we repeat this argument and discretize the scenario, we end up with a division of each scan-line in the autostereogram into several disjoint subsets of pixels. These subsets encode all the depth information of the scene. For a random dot stereogram, to each of these sets is now assigned a random color, and all pixels belonging to a set are plotted in the same color to generate the final autostereogram.

In the remainder of this section we will deduce exact formulas to find corresponding pixels for two different projection models of the HVS.

2.1. Constant viewing direction

This is the most simplifying assumption which leads to equations which can be evaluated rapidly, but are a bit less accurate than the ones in the other method. The autostereogram is constructed in the image plane Π, which shall coincide with the plane the focus of the visual system is adjusted to. The point C of convergence of the optical axes lies behind Π, directly in between both eye points E_L and E_R, Fig. 2. We assume that each scene point is projected onto the image plane parallel to the optical axis, so each projection direction intersects Π at a constant angle φ. For any scene point P, let $z = z(P)$ be the *depth of* P denoting its distance to the image plane. Directly from Fig. 2 we read off that the projections $\mathbf{p}_L$ and $\mathbf{p}_R$ are a distance

$$d = \frac{2z}{\tan\varphi} \tag{1}$$

apart from each other.

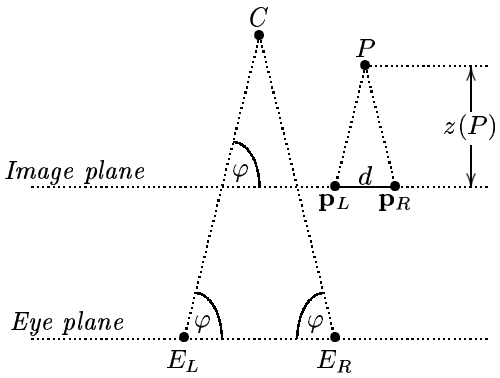

Figure 2. Relationship between two pixels with a constant viewing direction assumed for each scene point P.

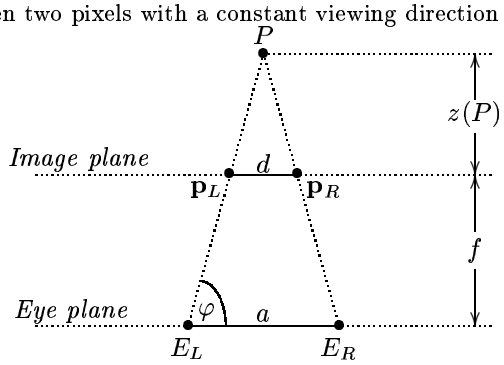

Figure 3. Relationship between two pixels with a constant focal length and eye distance assumed.

2.2. Constant eye distance and focal length

In this case we do not assume a constant angle φ, but instead make the more plausible assumption that the distance a between the eye points and the focal length f, denoting the distance from the image to the eye plane, is constant. The simplification we introduce is the assumption that the projected scene point P is centered in front of the eyes, Fig. 3. Again from simple geometric arguments we deduce

$$\tan\varphi \;=\; \frac{2}{a}(z+f), \tag{2}$$

yielding the distance

$$d \;=\; \frac{az}{z+f}. \tag{3}$$

3. CONSTRUCTING AN AUTOSTEREOGRAM

Our algorithm computes the autostereogram of an arbritrary geometry model. An initial image must be supplied that serves as the source pattern for the autostereogram generation. For a pure "random dot" stereogram, the pattern is initialized randomly, but other patterns can be used as well. For a convincing 3D impression of the autostereogram, the maximum depth of the scene should be bounded by a background plane. Therefore, a plane covering the entire visible region of the scene is placed at a certain distance behind the geometry.

Starting with the input image, the autostereogram algorithm transforms the input pattern iteratively until the final autostereogram is constructed. Starting from the left side of the pattern, in each step, the autostereogram properties as defined in Sect. 2 are enforced in subsequent vertical image strips. The algorithm terminates when the entire image plane is covered.

In each iteration step, the image is projected from the left-eye position onto the scene geometry, and reprojected into the right eye's view. This yields an adjacent vertical image strip of the autostereogram. These strips are invariant under subsequent iteration steps. The depth range of the scene must have a minimum non-zero distance from the image plane. It follows that the minimum displacement δ of a scene point is non-zero as well.

This means that every autostereogram image point influences only pixels having a distance greater δ. Therefore, in every iteration step, the newly generated vertical autostereogram strip has a minimum width of δ, so the algorithm terminates with a valid autostereogram after a finite number of steps.

More technically, for each iteration, the input pattern is used as a texture. The right-eye's view of the scene is rendered into the framebuffer, and afterwards the new image is copied back into the texture for the next iteration step. To perform the projection, for each vertex v of the geometry the distance $d(v)$ between the two corresponding eye-projection points in the autostereogram is calculated. The vertex's horizontal position is translated to the right by a shift of $d(v)/2$. The texture coordinates are used to take into account the projection of the image onto the scene as seen from the left eye. Thus, they are set to the same y position as the vertex, and the x position is set to the original vertex position shifted by $d(v)/2$ to the left. The right shift of the geometry enables the use of the graphics Z-Buffer to take into account the visibility due to occluded object regions.

Depending on the model for the HVS, the calculated distance $d(v)$ can either be linear in z, Eq. (1), or non-linear, Eq. (3). While the depth interpolation between two vertices is linear, due to the graphics card's texture coordinate interpolation, if small primitives are used to build up the geometry, the piecewise linearity is negligible.

The difference between the minimum and the maximum distance of two connected image points of an autostereogram corresponds to the minimum and maximum displayable depth in the scene. These distances are rounded to the nearest pixel position on the screen. Therefore, only a small number of discrete depth values can be visualized. As a result, discontinous depth steps are visible. To overcome this, texture filtering can be used to smooth the depth resolution. According to the real value of the calculated texture coordinates, the texture image is interpolated appropriately. The depth resolution then appears smooth, at the price of some blurring in the image pattern.

4. HARDWARE-ACCELERATED RENDERING

The algorithm described in the last section can be implemented straightforwardly using almost any kind of API for 3D-graphics. First, the frame buffer is initialized with random values. The number k of required iterations is known from the scene geometry. In each iteration, the frame buffer needs to be copied to a texture. With some APIs it is possible to render directly into a texture, which can be exploited to increase performance by removing the need for this copy operation. All projections from scene points P onto texture or image coordinates $\mathbf{p}_L$ or $\mathbf{p}_R$, respectively, can be performed very efficiently by customizable vertex transformations available on modern graphics hardware. Fig. 4 gives an example implementation of the transformation in Sect. 2.2 as a vertex program for nVidia's OpenGL extension `NV_vertex_program`. The DirectX version looks quite similar. After enabling this custom transformation and the current frame as a single texture for hardware texturing, arbitrary geometry can be rendered using the usual OpenGL primitives. One has to take care that the Z-Buffer is enabled in order to properly take into account visibility.

There is a simple modification of the original algorithm which aims at improving the quality of the final autostereogram. Since the original random texture strip on the left side of the image is stretched and compressed when the algorithm propagates it to the right during the iterations, it is slightly blurred due to texture filtering operations. This undesired effect can be reduced by starting in the center of the autostereogram and projecting into the left as well as into the right direction in the respective halves of the image. Interesting visual results can also be achieved by introducing some pre-defined patterns into the initial noise field, as proposed by Ninio and Herlin.[9]

5. RESULTS

Our hardware-accelerated implementation of the autostereogram rendering algorithm renders a complex scene consisting of 10,000 triangles at a resolution of 512×512 with a real-time frame rate of 25.6 Hz. More detailed information on the performance can be found in Table 1. Since the vertex projection described in Sect. 2.2 requires only two more instructions in a vertex program than the simple linear transformation, the impact on performance when using this more accurate method is negligible.

```
Input:    v[OPOS]   = Vertex world coordinates
          c[4]-c[7] = Model-view matrix
              c[0]  = Constant vector (0, 0, 0, 0)
              c[1]  = Constant vector (0.5, 0.5, 0.5, 0.5)
              c[2]  = Constant vector (0.5a, f, 0, 0)

Output:   o[HPOS]   = Homog. screen coordinates v_R
          o[TEX0]   = Texture coordinates v_L

  Compute V's camera coordinates in R0
   DP4 R0.x, c[4],v[OPOS]
   DP4 R0.y, c[5],v[OPOS]
   DP4 R0.z, c[6],v[OPOS]
   DP4 R0.w, c[7],v[OPOS]
  Product 0.5a · z in R1.x
   MUL R1.x, R0.z,c[2].x
  Quotient 1/(f + z) in R2.x
   ADD R2.x, R0.z,c[2].y
   RCP R2.x, R2.x
  Half disparity d/2 in R3.x
   MOV R3, c[0]
   MUL R3.x, R1.x,R2.x
  Compute new x-coordinate for v_L in R6
   ADD R6, R0,-R3
  Convert to the range [0, 1] for texture coordinates
   MAD o[TEX0], R6,c[1],c[1]
  Compute new x-coordinate for v_R
   ADD o[HPOS], R0,R3
```

Figure 4. Vertex program for the projection method in Sect. 2.2. For a vertex V in the input register `v[OPOS]` given in world coordinates, it computes both the texture coordinates $\mathbf{v}_L$ in the output register `o[TEX0]` as well as the vertex screen coordinates $\mathbf{v}_R$ in the output register `o[HPOS]`, respectively.

Resolution	# of triangles	k	Frame rate (Hz)
512×512	2,500	10	60.1
512×512	5,000	10	40.2
512×512	10,000	10	25.6
1024×512	1,000	20	26.2
1024×512	2,500	20	17.3
1024×512	5,000	20	12.4

Table 1. Dependency of frame rate on scene complexity and resolution. Pentium IV 1.8GHz, nVidia GeForce 4 Ti 4600.

Fig. 6 shows an example SIRDS rendered with our method. The propagation of the initialization texture starts in the middle of the image. A comparison between two autostereograms rendered either starting from the left or from the center is displayed in Fig. 7.

Subjective experiences of test observers indicate that a dynamic scene poses little problems to the viewer: The gaze remains correctly focused for the depth perception of a sequence of autostereograms. The fast change of patterns actually helps in keeping the eyes concentrating on the 3D object rather than the image texture. Since SIRDSs convey the true depth of objects, animated SIRDSs represent a useful and innovative approach to the interactive visualization of complex geometry. A demo movie is available from our web page,[10] as well as a Linux implementation of our method, which uses OpenGL and nVidia's extension `NV_vertex_program` to render the scenes.

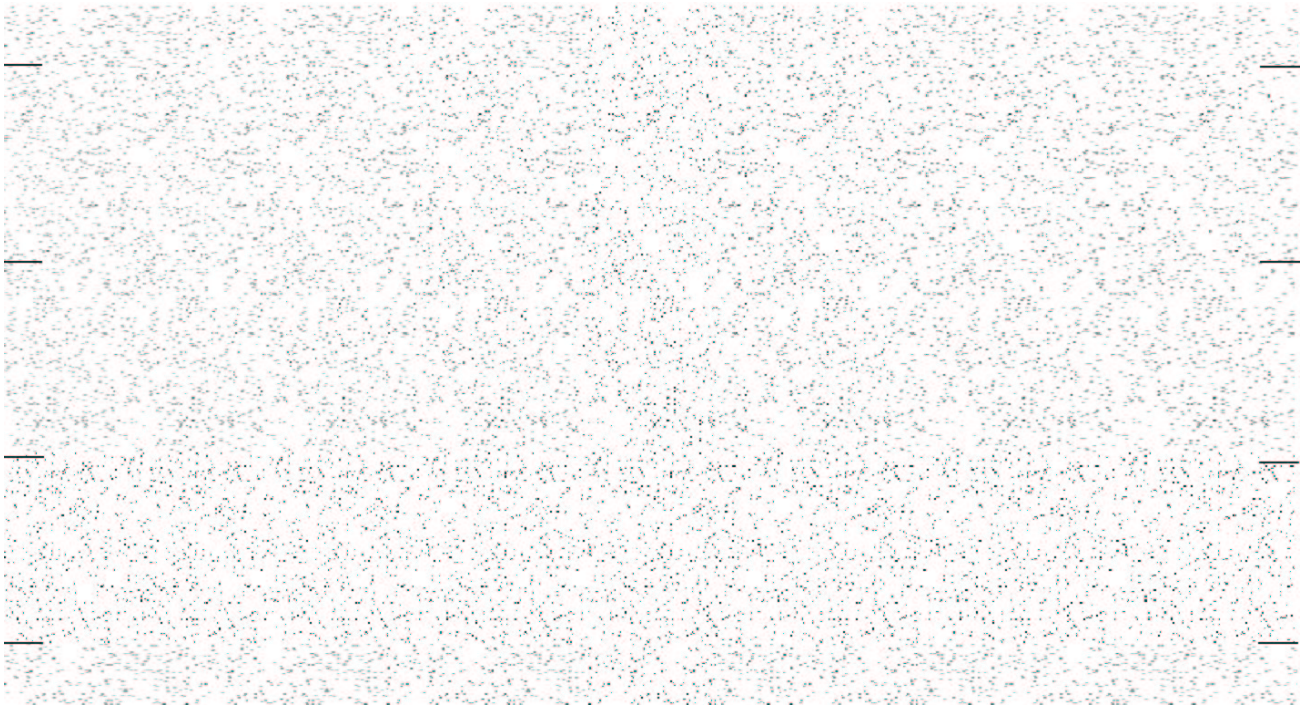

Figure 5. This autostereogram shows a ramp. On the left side it is close to the observer, the right side is receding away. The top third of the ramp is generated using a constant eye distance, the middle third a constant viewing direction. Both textures are interpolated to smooth the depth impression. The bottom third of the ramp shows the autostereogram without texture interpolation, discrete depth layers are visible.

6. CONCLUSIONS

We have presented a novel algorithm to create autostereograms which is very well suited for a hardware-accelerated implementation using off-the-shelf graphics hardware. It achieves real-time frame rates at high resolution for complex object geometry and represents an interesting and innovative way to visualize complicated 3D models conveying true depth information. There is no need for specialized, expensive hardware. Possible applications we will investigate in the future include a 3D mouse pointer to interact with the geometry, as well as tools for the interactive creation of artistic autostereogram images.

REFERENCES

1. B. Julesz and J. Miller, "Automatic stereoscopic presentation of functions of two variables," *Bell System Technical Journal* **41**, pp. 663–676, Mar. 1962.
2. B. Julesz, *Principles of Cyclopean Perception*, MIT Press, 1972.
3. C. Tyler and M. Clarke, "The autostereogram," *SPIE Stereoscopic Displays and Applications* **1258**, pp. 182–196, 1990.
4. D. Marr and T. Poggio, "Cooperative computation of stereo disparity," *Science* **194**, pp. 283–287, Oct. 1976.
5. H. Thimbleby, S. Inglis, and I. Witten, "Displaying 3D images: Algorithms for single-image random-dot stereograms," *IEEE Computer* **27**(10), pp. 38–48, 1994.
6. "Autostereogram history, FAQ and other internet resources." http://www.cs.waikato.ac.nz/~singlis/sirds.html.
7. *Magic Eye: A New Way of Looking at the World*, Andrews and McKeel, A Universal Press Syndicate Company, Kansas City, USA, 1993.
8. *Stereogram*, Cadence Books, San Francisco, USA, 1994.

9. J. Ninio and I. Herlin, "Speed and accuracy of 3D interpretation of linear stereograms," *Vision Research* **28**(11), pp. 1223–1233, 1988.
10. "The author's web page with demo videos and source code." http://www.mpi-sb.mpg.de/~petz/sirds.html.

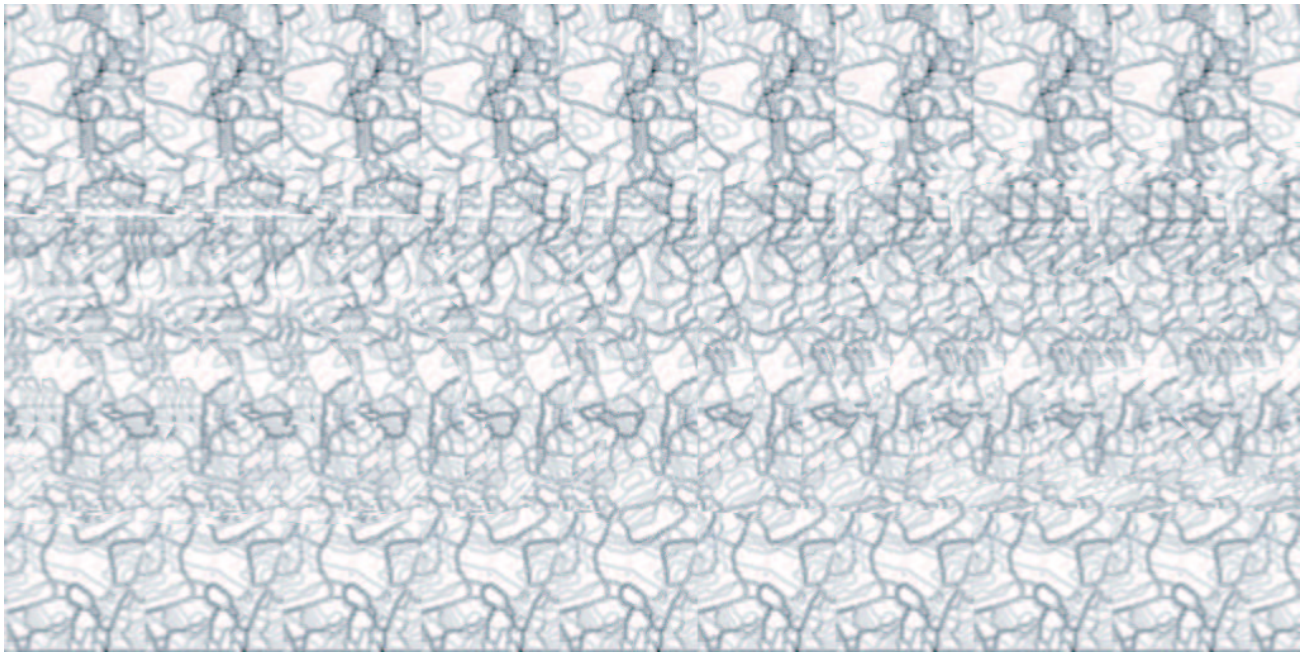

Figure 6. Autostereogram rendered using our hardware-accelerated implementation. Iteration starts in the center of the image with an initialization texture instead of a random color field. The frame rate for this 1024×512 pixel image is 29.5 Hz.

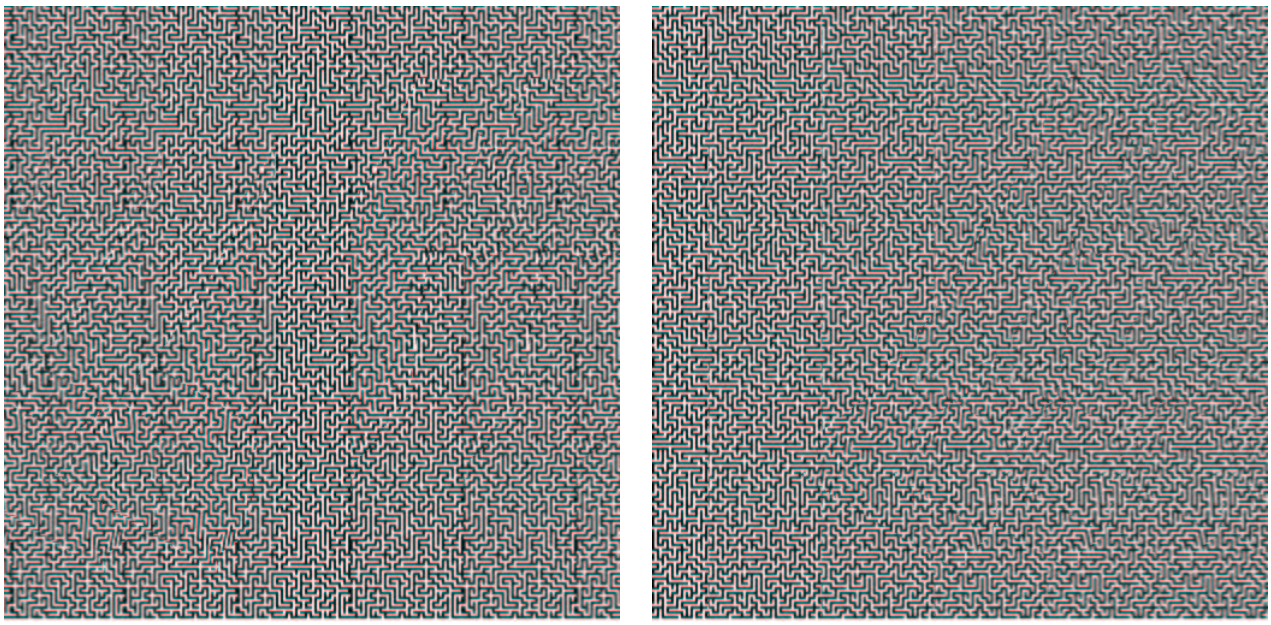

Figure 7. Comparison of two autostereograms with rendering starting either from the center (left image) or from the left border (right image). Note that the sharpness degrades visibly when the iteration proceeds further away from the initial stripe, so the modified method is preferable.