The Journal of the Society for Art and Science, Vol. 15, No. 2, pp. 98 – 110 (2016)

Real-Time Rendering Technique for Visual Expression of Arbitrary-Shaped Energy Wave

Taichi Watanabe[1)] **Masaki Abe**[1)] **Kouichi Konno**[2)]

1) Tokyo University of Technology 2) Iwate University

earth (at) gamescience.jp

Abstract

A visual expression called *Energy-Wave*, which shows light emission from high-energy fields, is popularly used in creative contents such as animations and computer games. This research proposes a new method for representing energy-Wave in real-time 3D graphics. Conventional energy-wave rendering techniques use the animation texture or the particle animation. However, the animation-texture method restricts the view point, and the boundaries of the energy-wave appear too clearly. The particle-animation method is not suitable for showing a dense energy field and not appropriate for covering a wide region. This paper presents a new method in which the energy distribution is described as a continuous scalar function. The new method is advantageous over the conventional methods since there is no restriction of the view point, resolution, and the covering region. The proposed method also utilizes GPGPU, and the output images are dynamically created from small input and can quickly deform the shape of the energy wave. The previous work of this research was able to deal with point-centered and line-centered energy-distribution functions. The new method can deal with torus-shaped and quadratic curve-centered distribution functions quickly.

1. Introduction

Visual expression of strong light emission from a high-energy blob moving in the 3D space is introduced to create impressive contents such as animations and cartoons. These expressions come from a special effect of the light ray used in the movies, which evolved in the Japanese animation. It is recently called *Energy-Wave* among animation-enthusiasts. Although this technique may be called differently in the different community, the expression itself is commonly used. (Nowrouzezahrai [1] calls this effect as "Artistic Volumetric Light.") This paper refers this visual effect as Energy-Wave. Our research defines the Energy-wave as a virtual phenomenon of strong light emission from high energy-density locations, which is a similar definition as in [2], and the shape of the light source deforms as the energy-distribution changes over time.

Visual effects using the energy-wave are widely used in variety of contents including animations, live-action films, computer graphics, and so on. The recent fast computer-graphics hardware allows real-time rendering of the energy-wave, and it is used in the interactive contents such as computer games. However, real-time graphics systems are typically optimized for drawing planes and curved surfaces, and not for the energy distribution in space. A computer-game program typically draws the energy-wave by the animation texture or the particle animation [3]. One of the advantages of the animation-texture method is that the artist can prepare the exact image of the energy-wave. On the other hand, the boundary between the inside and the outside of the energy-wave becomes unrealistically visible. This artifact can be made less visible by adjusting the transparency, in which case the view point is restricted. The particle-animation method does not restrict the view point. However, it is not good at drawing the energy-wave from the energy densely filling the large volume since it draws the energy-wave as a set of grainy point light sources.

Nowrouzezahrai [1] proposes a volumetric light-rendering technique based on the photon-mapping algorithm, which is suitable for dense energy distribution. However, this technique is for pre-rendered contents such as movies, and is not appropriate for real-time rendering.

Numerous researches have been presented methods for rendering dense energy distribution over the space. One of the most popular methods is a variation of the voxel-based method presented by Drebin et al. [4]. GPU-based volume-rendering techniques have been proposed recently [5][6], which can draw such a visual effect very fast. On the other hand, the energy distribution stored in the GPU memory space cannot be updated quickly, and it is not appropriate for covering large volume either.

Another class of techniques is based on implicit functions. Blinn [7] proposes "Blobby-Model", and Nishimura [8] proposes "Meta-Ball". Kanamori [9] and Kanai [10] proposes a technique to implement in the GPU. These methods are more focused on drawing a specific shape, and cannot directly be applied for rendering the energy-wave. Pasko [11], Schmitt [12] and Børlum [13] can express transparency and intensity. These methods are, however, for simulating the optical phenomenon of the incoming light within the geometric shape, and not suitable for the energy-wave expression in which the space itself emits the light.

Billeter et al. [14][15] propose a real-time rendering technique of the Tyndall-scattering effect by applying the real-time shadow rendering technique. This method is also for simulating the optical phenomenon inside the liquid and not suitable for the energy-wave.

This research is specialized for the energy-wave expression, similar to Abe et al. [2], and proposes new dense energy-distribution functions defined over the 3D space, and a real-time rendering technique of such functions. This research proposes two new energy-distribution functions, (i) torus shaped, and (ii) centered along arbitrary quadratic curve, in addition to a point-centered (spherical) and a line-centered (cylindrical) functions. Arbitrary quadratic curves include Bézier curves and parabolic curves, which substantially expands the freedom of the modeling of the energy distribution. Another advantage of this method is that it can deform the shape of the energy wave quickly because this method ren-

ders the energy-wave in each frame, as opposed to the many previous methods that require a pre-rendered image.

In the rest of the paper, Section 2 describes torus-shaped and quadratic curve-centered energy-distribution functions. Sections 3 and 4 explains the proposed fast-rendering technique using GPGPU. Section 5 shows some experimental results, which confirms that the proposed method can achieve practical rendering speed.

2. Energy-Wave Distribution Function

This method describes the energy-intensity at a point in the three-dimensional space as the following scalar function:

$$\psi(\mathbf{P}) : \ \mathbf{P} \in \mathbb{R}^3, \ \psi(\mathbf{P}) \in \mathbb{R}, \tag{1}$$

where $\mathbb{R}$ is the set of real number. The higher the intensity value $\psi(\mathbf{P})$ the brighter the light emission at $\mathbf{P}$. The next section describes details of the light-emission calculation.

2.1. Energy-Distribution Function of the Previous Method

In our previous work [2], a distribution function $S(\mathbf{P}, \mathbf{M})$ that decays radially from an arbitrary point $\mathbf{M}$ is defined as equation (2):

$$S(\mathbf{P}, \mathbf{M}) = \frac{1}{|\mathbf{P} - \mathbf{M}|}. \tag{2}$$

Also a distribution function that decays radially from a line that passes through point $\mathbf{M}$ and is parallel to unit vector $\mathbf{D}$ as equation (3):

$$C(\mathbf{P}, \mathbf{M}, \mathbf{D}) = \frac{1}{\sqrt{|\mathbf{P} - \mathbf{M}|^2 - ((\mathbf{P} - \mathbf{M}) \cdot \mathbf{D})^2}}, \tag{3}$$

where $\mathbf{P}$ is a point in the space.

2.2. Torus-Shaped Distribution Function

A general torus surface is defined as a surface of revolution of a circle of radius r lying on the xz-plane:

$$C : (x - R)^2 + z^2 = r^2 \quad (0 < r < R) \tag{4}$$

revolved about the z-axis. The following equation (5) describes its implicit form:

$$T : \left(\sqrt{x^2 + y^2} - R\right)^2 + z^2 = r^2. \tag{5}$$

Solving equation (5) for r yields the following equation (6):

$$r = \sqrt{\left(\sqrt{x^2 + y^2} - R\right)^2 + z^2}. \tag{6}$$

This defines distance between a point (x, y, z) and the core curve, or major circle, of the torus. From this equation, the torus-shaped energy-distribution function can be defined as equation (7):

$$T(\mathbf{P}, R) = \frac{1}{\sqrt{\left(\sqrt{{P_x}^2 + {P_y}^2} - R\right)^2 + {P_z}^2}}. \tag{7}$$

The major-radius of the torus can be controlled by adjusting the value of R.

In this method, the energy intensity at a certain point is influenced only by the distance from the nearest point on the circular light-source arc. It is also possible to define an energy-distribution function in which the energy intensity is influenced by the multiple points on the arc. Although such a function may appear to be mathematically correct, it yields an unintentional outcome. Since entire arc contributes to the energy intensity near the center of the circle, the energy distribution becomes more like an ellipsoid than a torus, unless the circle radius is sufficiently large. Therefore, this method uses an energy-distribution function in which the energy intensity is a function of distance from the nearest point on the arc. Figure 1 shows a comparison between two types of energy-distribution functions, an image rendered with the energy-distribution function of this method (left), and with the energy-distribution function in which multiple points on the arc contribute to the energy-intensity at a point (right).

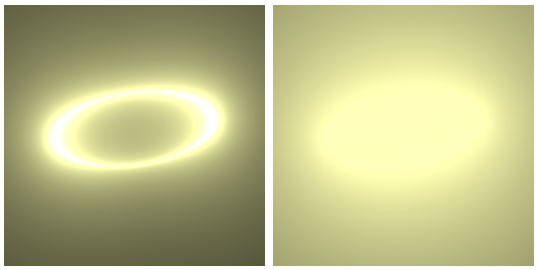


Figure 1: An image rendered with the energy-distribution function of this method (left), and with the energy-distribution function in which multiple points on the arc contribute to the energy-intensity at a point (right).

2.3. Energy-Distribution Function Centered along Quadratic Parametric Curve

This section describes an energy-distribution function centered along a quadratic parametric curve, which can be described by a sequence of 3D vectors $\{\mathbf{C}_0, \mathbf{C}_1, \mathbf{C}_2\}$ and a parameter t ($t \in \mathbb{R}$) as follows:

$$\mathbf{C} : \mathbf{C}(t) = \mathbf{C}_2 t^2 + \mathbf{C}_1 t + \mathbf{C}_0 \ . \tag{8}$$

Although this expression cannot represent arbitrary quadratic curves, such as circular arcs and hyperbolic curves, it can describe parabolic curves and free-form curves known as a Bézier curve. It also can describe a free form curve known as Bézier curve. A quadratic Bézier curve defined by three control points $\{\mathbf{B}_0, \mathbf{B}_1, \mathbf{B}_2\}$:

$$\mathbf{C}(t) = (1-t)^2\mathbf{B}_0 + 2t(1-t)\mathbf{B}_1 + t^2\mathbf{B}_2 \tag{9}$$

can be re-written in the form of equation (8) by substituting:

$$\begin{cases} \mathbf{C}_2 = \mathbf{B}_0 - 2\mathbf{B}_1 + \mathbf{B}_2 \\ \mathbf{C}_1 = -2\mathbf{B}_0 + 2\mathbf{B}_1 \\ \mathbf{C}_0 = \mathbf{B}_0 \end{cases} . \tag{10}$$

If $\mathbf{C}(\alpha)$ is a point on a C^1-continuous curve $\mathbf{C}$ and is closest to point $\mathbf{P}$, a line passes through $\mathbf{P}$ and $\mathbf{C}(\alpha)$ and the tangent of curve $\mathbf{C}$ at $\mathbf{C}(\alpha)$ are perpendicular as shown in Figure 2.

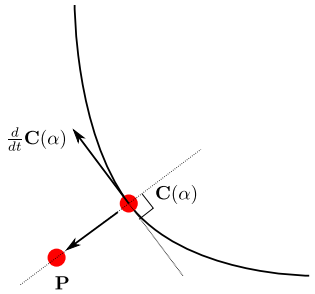


Figure 2: Calculation of the Closest Point.

This implies equation (11):

$$\frac{d}{dt}\mathbf{C}(\alpha) \cdot (\mathbf{P} - \mathbf{C}(\alpha)) = 0. \tag{11}$$

Therefore, real solutions of equation (11) give candidates of the closest point.

Substituting equation (11) into (8) yields the following cubic equation:

$$\begin{aligned} &\frac{d}{dt}\mathbf{C}(t) \cdot (\mathbf{P} - \mathbf{C}(t)) = \\ &\quad 2|\mathbf{C}_2|^2\, t^3 + 3(\mathbf{C}_1 \cdot \mathbf{C}_2)\, t^2 + \\ &\quad \left(2(\mathbf{C}_0 \cdot \mathbf{C}_2) + |\mathbf{C}_1|^2\right)\, t + \mathbf{C}_1 \cdot (\mathbf{C}_0 - \mathbf{P}). \end{aligned} \tag{12}$$

A cubic equation can be analytically solved and does not require a numerical method for finding exact solutions.

The proposed method calculates the distance between point $\mathbf{P}$ and curve $\mathbf{C}$ in the following steps:

1. Solve equation (12) and find real solutions that are within the parameter domain of the curve (eg. $0 \le t_i \le 1$ if the curve is a Bézier curve,) which are denoted as $\{t_1, t_2, t_3\}$. Solutions t_2 and t_3 may not exist.

2. For each t_i calculated in Step 1, calculate a point on the curve $\mathbf{C}(t_i)$.

3. Calculate two end points of the curve (eg. $\mathbf{C}(0)$ and $\mathbf{C}(1)$ if the curve is a Bezier curve).

4. Calculate distances between $\mathbf{P}$ and each point calculated in Steps 2 and 3.

5. Minimum of the distances calculated in Step 4 is taken as the distance between $\mathbf{P}$ and $\mathbf{C}$.

The proposed method employs Cardano's formula [16] for finding solutions of the cubic equation. Details is described in Appendix A.

The energy-distribution function $Q(\mathbf{P}, \mathbf{C})$ can then defined as:

$$Q(\mathbf{P}, \mathbf{C}) = \frac{1}{L(\mathbf{P}, \mathbf{C})}, \tag{13}$$

where $L(\mathbf{P}, \mathbf{C})$ is the distance between point $\mathbf{P}$ and curve $\mathbf{C}$ calculated in the above steps.

Similar to the torus-shaped energy distribution function described in Section 2.2, this method only considers the distance from the nearest point on the curve for calculating the energy intensity at a certain point. An energy-distribution function in which multiple points on the curve influence the energy intensity at a point has two major problems:

(1) Energy intensity is lower near the end points of the curve.

(2) Energy intensity inside of the high-curvature segment of the curve becomes higher than desired.

The shape of the energy-wave does not look similar to the input curve as a result of the two problems. In other words, it becomes difficult for the user to predict the shape of the energy wave from the input curve and parameters. Figure 3 shows the comparison between the energy-distribution function of the proposed method (left), and the energy-distribution function in which multiple points contribute to the energy-intensity at a point (right).

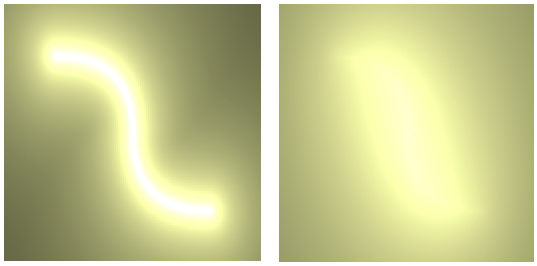


Figure 3: An image rendered with the energy-distribution function in which the intensity is a function of the distance from the nearest point on the arc (left), and with a function in which multiple points contribute to the intensity at a point (right).

The image on the left of Figure 3 shows the energy-wave that appears to be similar to the input curve. On the other hand, the energy-distribution on the right shrinks near the both ends of the curve. Also it swells into the inside of the curvature. We believe the image on the left better expresses the intention of the user, and therefore the proposed method uses the energy-distribution function in which the energy-intensity at a point is a function of the distance from the nearest point on the curve.

2.4. Locally Adjusting Intensity by a B-Spline Basis Function

The function $Q(\mathbf{P}, \mathbf{C})$ described in Section 2.3 uniformly distributes energy over the curve. If it is possible to control the energy distribution locally for the different parts of the curve, such functionality increases the flexibility of the expression. For example, it can easily implement an expression of the energy wave travelling along the curve. Figure 2 shows an example of the local adjustment.

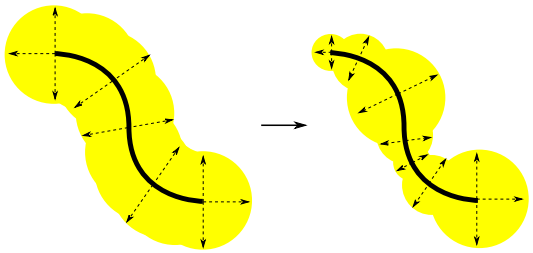


Figure 4: Example of the local adjustment.

Earlier work [2] theoretically can generate a similar expression by combining spherical and cylindrical distribution functions. However, the earlier work requires evaluation of multiple distribution functions, which yield slower rendering. The new method requires only one distribution function, and therefore can achieve such a flexible expression without sacrificing the rendering speed.

The proposed method uses a B-Spline basis function for controlling the energy distribution locally. The advantages of using a B-Spline basis function are:

1. It can express a natural appearance of the energy distribution because continuity of the function has been very well studied.

2. It can be evaluated quickly since it takes only four arithmetic operators of real numbers.

3. It is more flexible than Bezier basis function in terms of local controllability.

A B-Spline basis function is defined by a monotonically increasing column
$U = [u_0, u_1, \ldots, u_{m-1}]$ and degree k as:

$$N_i^0(t) = \begin{cases} 1 & if\ u_i \le t \le u_{i+1} \\ 0 & else \end{cases},$$
$$N_i^k(t) = \frac{t - u_i}{u_{i+k} - u_i} N_i^{k-1}(t) - \frac{t - u_{i+k+1}}{u_{i+k+1} - u_{i+1}} N_{i+1}^{k-1}(t), \quad (14)$$

where U is called knot vector.

An adjustment function $S(t)$ is defined with a series $W = [w_0, w_1, \ldots, w_{n-1}]$ as:

$$S(t) = \sum_{i=0}^{n-1} w_i N_i^k(t), \quad (15)$$

where W is a series that defines local intensity over the curve.

In this research, we have selected $k = 2$ for faster rendering and limiting influence of w_i within the immediate-neighbor segment.

The length of W is variable. When the number of elements in W is n, and the curve is parameterized for the range $[0, 1]$, w_i controls an energy intensity of the segment around the location where the parameter is $\frac{i}{n-1}$. The knot vector for n should be defined as:

$$U = \left[0,\ 0,\ 0,\ \frac{1}{n-2}, \frac{2}{n-2}, \ldots,\ \frac{n-3}{n-2},\ 1,\ 1,\ 1\right]. \quad (16)$$

For example, Figure 5 shows the plot of $S(t)$ when $W = [1.0, 0.6, 0.2, 0.4, 0.8, 0.6, 1.0]$.

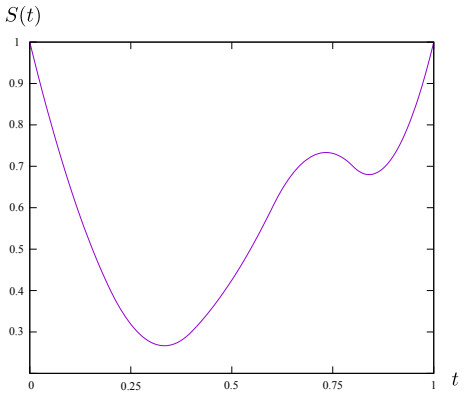


Figure 5: Graph of $S(t)$.

Finally, $S(t)$ is combined with the function for locally adjusting the energy intensity (13), and the new distribution function is defined as:

$$B(\mathbf{P}, \mathbf{C}) = S(t) \cdot Q(\mathbf{P}, \mathbf{C}), \quad (17)$$

where t is the parameter of curve $\mathbf{C}$ at the nearest point to point $\mathbf{P}$.

2.5. Multiple Light Sources

The final scalar field function $\psi(\mathbf{P})$ is the sum of all energy-distribution functions defined by equations (2), (3), (7), and (13) as:

$$\psi(\mathbf{P}) = \sum_i s_i S_i(\mathbf{P}) + \sum_i c_i C_i(\mathbf{P}) + \sum_i t_i T_i(\mathbf{P}) + \sum_i b_i B_i(\mathbf{P}), \quad (18)$$

where s_i, c_i, t_i, and b_i are real coefficients for each energy-distribution function, and are called *energy coefficients* in the rest of the paper. The effect of each energy-distribution function can be controlled by adjusting these coefficients.

3. Rendering

The proposed rendering method is similar to the conventional ray-casting method. The ray-casting method shoots a ray per pixel from the view point toward the projection plane, and then numerically or analytically integrates the volume data along the ray to decide the color of the pixel.

Since the volume data in the proposed method is a scalar function, instead of a voxel data, the

pixel color is decided based on the curvilinear integration of the scalar function.

3.1. Deciding the Interval of Integration

Let **S** be an arbitrary position in the pixel space on the projection plane, and **E** be a point that lies on the far extension of the line on which **S** and the viewing (or camera) position are lying.

The integration interval is the line segment between **S** and **E**. This line segment L is described using parameter s as:

$$L : \mathbf{L}(s) = (1-s)\mathbf{S} + s\mathbf{E} \qquad (0 \leq s \leq 1). \quad (19)$$

E lies on the extension of the viewing direction from **S**, and must be placed certain distance away from **S**. E must also be sufficiently distant from the energy wave. However, the calculation becomes less precise as **E** moves farther away from **S**, and may yield an inappropriate rendering. In the experiments performed in this research, the distance between the energy wave and E needs to be approximately 4 times of the distance between the projection plane and the energy-wave center to obtain a good result.

3.2. Calculation of the Brightness

The brightness of the pixel is calculated as equation (20):

$$\int_0^1 \psi\left(\mathbf{L}(s)\right)\zeta(s)ds, \quad (20)$$

where ψ is a scalar field function of the energy distribution in equation (18), and ζ is the decay function of distance. In general, the light intensity at the view point is inversely proportional to the distance square, and ζ simulates this effect. The following decay function is used in this research:

$$\zeta(s) = \frac{\alpha}{(s+\beta)^2}, \quad (21)$$

where α and β are the constants for making adjustments.

It is ideal if equation (20) can be analytically calculated and exact value of the light intensity can be obtained. However, the analytical solution to equation (20) is a non-elementary function that consists of hundreds of terms, numerical calculation is thus more practical. This method first defines $f(s)$ as:

$$\psi\left(\mathbf{L}(s)\right)\zeta(s) = f(s), \quad (22)$$

and applied Simpson's formula to numerically approximate the solution to equation (20) as:

$$\int_0^1 f(s)ds \approx \frac{1}{3n}\left(f(0) + 2\sum_{j=1}^{\frac{n}{2}-1} f\left(\frac{2j}{n}\right) + 4\sum_{j=1}^{\frac{n}{2}} f\left(\frac{2j-1}{n}\right) + f(1)\right), \quad (23)$$

where n is the division count. The larger n will increase the accuracy. The experimental results show that $n = 200$ yields adequate picture quality. Figure 6 is a schematic drawing of the above calculations.

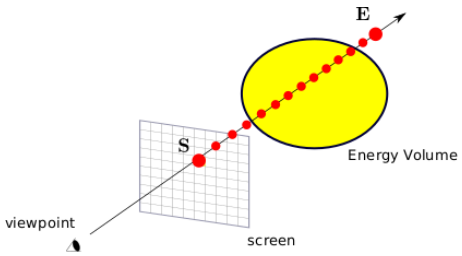


Figure 6: Schematic drawing of the brightness calculation.

4. Parallel Processing with GPGPU

This research has accelerated the processing by the GPGPU (General Purpose Graphics Processing Unit) technique, in which the GPU (Graphics Processing Unit) performs general computations that are traditionally handled by a CPU. The proposed method calculates equation (20) for each pixel, and it is virtually impossible to render the scene real-time with a CPU. Instead, equation (20) can be made a process that is assigned to a processing unit of a GPU, and the calculation can be substantially accelerated.

There are multiple options to utilize GPGPU processing such as

- shader languages including HLSL and GLSL,
- CUDA language proposed by NVIDIA,
- OpenCL proposed by Khronos Group.

Shader languages are specialized for drawing and the volume data needs to be converted to a texture, which imposes a substantial restriction in the resolution. Although CUDA is very popular, it can be used only on the GPUs manufactured by NVIDIA. Considering these limitations, we have chosen OpenCL for the experimental implementation.

5. Validation

The proposed method has been implemented and tested on a PC shown in Table 1.

Table 1: The PC configuration used for the validation.

Machine	MacPro (Late 2013)
CPU	3.7Ghz Quad-Core Intel Xeon E5
Memory	16GB 1866Mhz DDR3 ECC
GPU	AMD FirePro D500 3GB Memory

FirePro D500 consists of 1,526 stream processors.

The basic development and validation are performed on MacOSX with a GPU manufactured by AMD. The same implementation has been tested on Windows 7 with a GPU manufactured by NVIDIA, which showed a similar performance.

5.1. Speed performance

Tables 2 and 3 show a comparison of the speed performance for different resolutions and sampling division counts. We have tested with three different resolutions, 128x128, 256x256 and 512x512, and two different sampling division count, 64, 128 and 256. We have measured the frames per second under these conditions.

Table 2: Speed performance result (Torus shaped), all values given in frames per second.

resolution	64 div	128 div	256 div
128x128	384.61	168.63	62.35
256x256	116.69	46.49	16.85
512x512	31.23	13.49	5.21

Table 3: Speed performance result (Bézier curve), all values given in frames per second.

resolution	64 div	128 div	256 div
128x128	116.69	60.17	28.94
256x256	34.64	17.16	8.31
512x512	9.00	4.40	2.15

Real-time rendering requires at least 60 FPS in general. The results indicate that 256x256 resolution for torus-shaped energy field, and 128x128 resolution for Bézier curve-centered energy field are practical. Although these resolutions are smaller compared to the recent PC monitors, the methods for interpolating a low-resolution texture for high-definition rendering have been developed [17]. Such techniques work particularly well in the image that the color does not change drastically within neighboring pixels, which is exactly the output of the proposed method. With such an interpolation scheme, the proposed method can be used for practical purposes.

5.2. Output images

This section presents some examples of the output images. All images are created with 256x256 resolution.

Figures 7 and 8 show a torus-shaped energy distribution rendered from three different angles. Parameter R of equation (7) and energy coefficient for the torus-shaped energy distribution t_i of equation (18) are also shown in the caption.

Figure 7: Images of a torus-shaped energy distribution. ($R = 0.9, t_i = 5.1$)

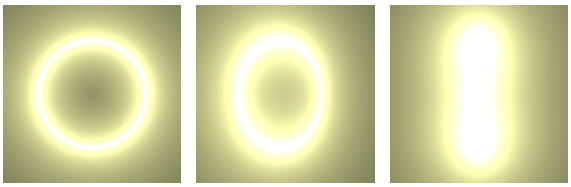


Figure 8: Images of a torus-shaped energy distribution. ($R = 3.0, t_i = 7.4$)

Figures 9 and 10 are the images of the energy distribution defined by two smoothly-connected Bézier curves rendered from three different angles. Energy coefficient for the curve-shaped energy distribution b_i of equation (18) is also shown in the caption.

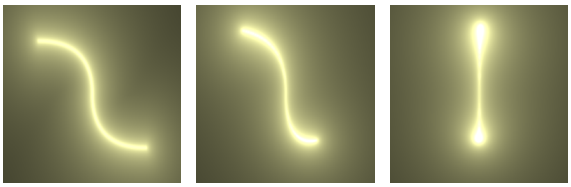


Figure 9: Images of a Bézier curve-centered energy distribution. ($b_i = 5.0$)

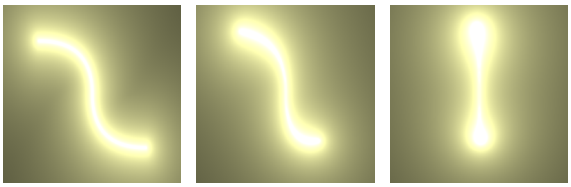


Figure 10: Images of a Bézier curve-centered energy distribution. ($b_i = 7.5$)

Figure 11 is an image after dynamically moving control points of the Bézier curve from the state shown in Figure 9.

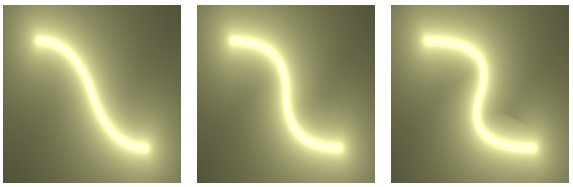


Figure 11: Images of a Bézier curve after dynamically moving control points.

Since the proposed method re-builds the image based on the view point and parameters in every frame, change of view point or parameters will not cost additional processing time.

Figure 12 shows an example of the local energy-distribution control explained in section 2.4. The left image of Figure 12 is rendered with maximum intensity uniformly over the curve, and the right image is rendered with the local-intensity series of $W = [1.0, 0.2, 0.6, 1.0, 0.5, 0.8, 1.0]$. Energy coefficient $b_i = 8.0$ is used for rendering the two images.

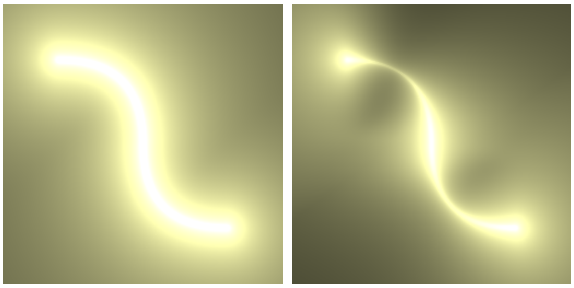


Figure 12: Rendering without local-intensity adjustment (left) and with adjustment (right).

Figure 13 shows an example of a practical application of the proposed method.

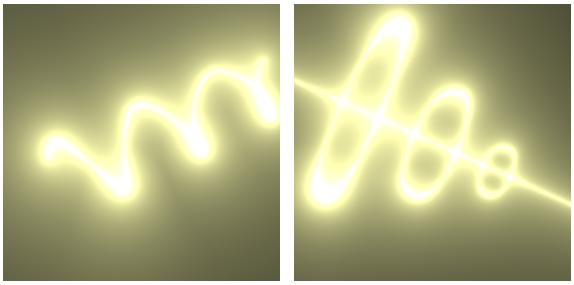
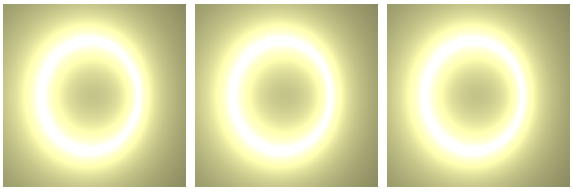


Figure 13: Sample output images of a practical application.

5.3. Effect of the division count in the numerical integration

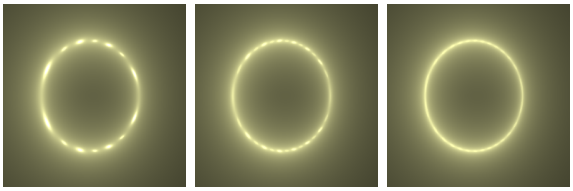
Since this method approximates the brightness by numerically integrating the energy field, low division count (or large integration step) may adversely affect the output image. In particular, when combined with the small coefficient of the energy-distribution function, the output image may show undesired unevenness.

Figures 14 through 17 are the images rendered with different division counts. Figures 14, 15, 16, and 17 are the images with a torus-shaped energy distribution with energy coefficient $t_i = 8.0$, a torus-shaped energy distribution with energy coefficient $t_i = 4.0$, a curve-shaped energy distribution with energy coefficient $b_i = 8.0$, and a curve-shaped energy distribution with energy coefficient $b_i = 4.0$, respectively. The division count is shown underneath each image.



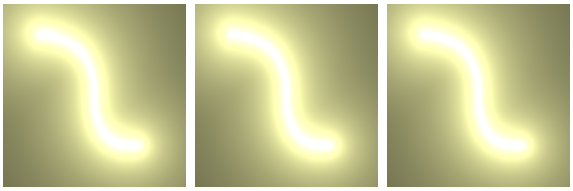
64 128 256

Figure 14: Torus-shaped energy distribution. ($t_i = 8.0$)



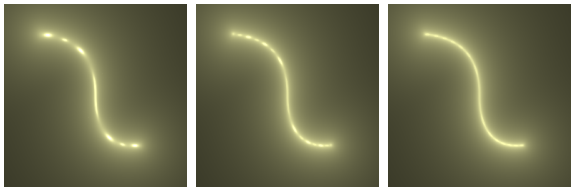
64 128 256

Figure 15: Torus-shaped energy distribution. ($t_i = 4.0$)



64 128 256

Figure 16: Curve-shaped energy distribution. ($b_i = 8.0$)



64 128 256

Figure 17: Curve-shaped energy distribution. ($b_i = 4.0$)

As can be seen in Figures 14 and 16, the difference in the rendering is almost unnoticeable when the energy coefficient is higher. However, as can be seen in Figures 15 and 17, undesired unevenness becomes more apparent when the division count is lower. Although such undesired unevenness can be avoided by choosing a higher division count, a higher division count requires more computation and reduces the frame rate. The division count thus must be carefully chosen when the energy coefficient is lower.

5.4. Discussions of potential improvement

As discussed in Section 5.3, the proposed method yields undesired unevenness when the low division count in the numerical integration and the low energy coefficient are combined. A stochastic sampling method [18] has been proposed for dealing with this type of periodic unevenness. The result of this method may be improved by applying this kind of technique. Studying the effect of such a technique is one of future research topics.

The proposed method tends to increase the brightness of the pixels when energy is distributed more uniformly along the view direction, even if the intensity of the energy is not very high, than when high energy is concentrated in one point. This gives unnatural impression when the view angle is rotated. This problem can potentially be corrected by applying a different decay function for equation (20). It is also one of the future research topics.

6. Conclusions

This research has expanded the authors' previous research [2] and has proposed a method for drawing the new forms of the energy wave. In

particular, the Bézier curve-centered energy field can visualize free-form energy waves, and the intensity over the curve can locally be controlled by the local-intensity series. These new features enable substantially more complex visual effects compared to the previous technique.

The current form of the method, however, has some obstacles to overcome to become practical. First of all, the current method requires the designer to specify numerical parameters for defining the energy distribution, which are not very intuitive. It is necessary to provide with a tool for editing the parameters and a graphical user interface for easily adjusting the parameters in a game engine. Also the hidden-surface removal, which was implemented in the previous method, has not been implemented and tested with the new method, which we are working on right now. Also, since the energy wave is a light emission from the energy field, the surrounding objects must reflect the light from the energy wave and should be rendered accordingly. By addressing these points, this method will substantially expand the graphical expressions in creative contents such as computer games.

References

[1] D. Nowrouzezahrai, J. Johnson, A. Selle, D. Lacewell, M. Kaschalk, and W. Jarosz. A programmable system for artistic volumetric lighting. *ACM Transactions on Graphics*, Vol. 30, No. 4, pp. 29:1–29:8, 2011.

[2] M. Abe and T. Watanabe. Real-time rendering of energy-wave expression. *The Journal of the Society for Art and Science*, Vol. 9, No. 3, pp. 93–101, 2010.

[3] M. Nito, T. Watanabe, M. Kakimoto, and K. Mikami. Energy-wave expression considering a collision in real-time 3DCG. *The Journal of the Society for Art and Science*, Vol. 13, No. 3, pp. 144–153, 2014.

[4] R. Drebin, L. Carpenter, and P. Hanrahan. Volume rendering. *Computer Graphics*, Vol. 22, No. 4, pp. 65–74, 1988.

[5] J. Krufer and R. Westermann. Acceleration techniques for GPU-based volume rendering. *Proceedings of the 14th IEEE Visualization*, pp. 287–292, 2003.

[6] J. Stuart, C. Chen, K. Ma, and J. Owens. Multi-GPU volume rendering using mapreduce. *Proceedings of the 19th ACM HDPC'10*, pp. 841–848, 2010.

[7] J. Blinn. A generalization of algebraic surface drawing. *ACM Transactions of Computer Graphics*, Vol. 1, No. 3, pp. 235–256, 1982.

[8] H. Nishimura, M. Hirai, T. Kawai, I. Shirakawa, and K. Omura. Object modeling by distribution function and a method of image generation. *Journal of papers given by at the Electronics Communication Conference*, Vol. 568, pp. 718–725, 1985.

[9] Y. Kanamori, Z. Szego, and T. Nishita. GPU-based fast ray casting for a large number of metaballs. *In Proc. Eurographics*, Vol. 27, No. 2, pp. 351–360, 2008.

[10] T. Kanai, Y. Ohtake, H. Kawai, and L. Kase. GPU-based rendering of sparse low-degree implicit surface. *In Proc. of GRAPHITE*, pp. 165–171, 2006.

[11] A. Pasko, V. Adzhiev, A. Sourin, and V. Savchenko. Function representation in geometric modelling: concept, implementation and applications. *The Visual Computer*, Vol. 11, No. 8, pp. 429–446, 1995.

[12] B. Schmitt, A. Pasko, and C. Schlick. Constructive hypervolume modeling using extended space mappings. Heterogeneous objects modelling and applications: collection of papers on foundations and practice, pp. 167–192. Springer-Verlag, Berlin, 2008.

[13] J. Børlum, B. B. Christensen, T. K. Kjeldsen, P. T. Mikkelsen, K. Ø . Noe, J. Rimestad, and J. Mosegaard. SSLPV: subsurface light propagation volumes. *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics*, pp. 7–14, 2011.

[14] M. Billeter, E. Sintorn, and U. Assarsson. Real time volumetric shadows using polygonal light volumes. *Proceedings of the Conference on High Performance Graphics*, pp. 39–45, 2010.

[15] M. Billeter, E. Sintorn, and U. Assarsson. Real-time multiple scattering using light propagation volumes. *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pp. 119–126, 2012.

[16] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, 1992.

[17] M. Pharr, R. Fernando, and T. Sweeney. *GPU Gems 2*. Addison-Wesley Professional, 2005.

[18] Robert L. Cook. Stochastic sampling in computer graphics. *ACM Transactions on Graphics*, Vol. 5, No. 1, pp. 51–72, 1986.

A. Real solutions of cubic equations

This section explains a method for calculating real solutions of cubic equations such as:

$$x^3 + ax^2 + bx + c = 0 \tag{24}$$

by Cardano's formula. The solution described here is the minimum required steps. Please refer to [16] for the details.

The first step is to calculate three real numbers p, q, and r as follows:

$$p = -\frac{a^2}{9} + \frac{b}{3}, \tag{25}$$

$$q = \frac{a^3}{27} - \frac{ab}{6} + \frac{c}{2}, \tag{26}$$

$$r = p^3 + q^2. \tag{27}$$

Then calculate two complex numbers u and v as follows:

$$u = \sqrt[3]{-q + \sqrt{r}}\,, \tag{28}$$

$$v = \sqrt[3]{-q - \sqrt{r}}. \tag{29}$$

When u and v are calculated programmatically, this step needs to be done differently depending on the sign of r.

- If r is zero or positive: u and v are both real numbers.
  1. Calculate $s = -q + \sqrt{r}$ and $u = \sqrt[3]{s}$ if $s \geq 0$, or $u = -\sqrt[3]{-s}$ if $s < 0$.
  2. Calculate $t = -q - \sqrt{r}$ and $v = \sqrt[3]{t}$ if $t \geq 0$, or $v = -\sqrt[3]{-t}$ if $t < 0$.
- If r is negative: u and v are both complex numbers.
  1. Let z a complex number defined as $z = -q + \sqrt{-r}\,i$.
  2. Calculate argument of z as $\theta = \arctan\left(\frac{q}{r}\right)$.
  3. Calculate $u = \sqrt[3]{z} = \sqrt[3]{|z|}\left(\cos\frac{\theta}{3} + i\sin\frac{\theta}{3}\right)$.
  4. Calculate $v = \overline{u}$.

Then calculate three complex numbers ω_1, ω_2 and ω_3 as:

$$\omega_1 = e^0 = 1, \tag{30}$$

$$\omega_2 = e^{\frac{2\pi i}{3}} = -\frac{1}{2} + \frac{\sqrt{3}}{2}\,i, \tag{31}$$

$$\omega_3 = e^{\frac{4\pi i}{3}} = -\frac{1}{2} - \frac{\sqrt{3}}{2}\,i. \tag{32}$$

Then calculate:

$$\begin{aligned} u_1 &= \omega_1 u, \quad u_2 = \omega_2 u, \quad u_3 = \omega_3 u, \\ v_1 &= \omega_1 v, \quad v_2 = \omega_2 v, \quad v_3 = \omega_3 v. \end{aligned} \tag{33}$$

Finally, the solutions to the cubic equation are three of the following nine numbers:

$$\begin{aligned} &u_1 + v_1, \quad u_1 + v_2, \quad u_1 + v_3, \\ &u_2 + v_1, \quad u_2 + v_2, \quad u_2 + v_3, \\ &u_3 + v_1, \quad u_3 + v_2, \quad u_3 + v_3. \end{aligned} \tag{34}$$

The proposed method only is concerned about the real solutions. If the cubic equation has real solutions, all of them match the real numbers among (34). Therefore, the purpose in the proposed method is satisfied by taking the numbers with zero imaginary part.

渡辺大地 (Taichi Watanabe)

1994 年慶応義塾大学環境情報学部卒業. 1996 年慶応義塾大学大学政策・メディア研究科修士課程修了. 修士 (政策・メディア). 1999 年より東京工科大学メディア学部講師. コンピューターグラフィックスやゲーム制作に関する研究に従事. 著書に「CG とゲームの技術」がある. 情報処理学会, 芸術科学会, 画像電子学会会員.

阿部雅樹 (Masaki Abe)

2008 年東京工科大学メディア学部卒業. 2010 年東京工科大学大学院バイオ・情報メディア研究科メディアサイエンス専攻修士課程修了. メディア学修士. 2016 年より東京工科大学メディア学部実験助手. コンピューターグラフィックスの研究に従事.

今野晃市 (Kouichi Konno)

1985 年, 筑波大学第三学群情報学類卒業. (株) リコーソフトウェア研究所, ラティス・テクノロジー (株) を経て, 現在, 岩手大学工学部教授. 著書に「3 次元形状処理入門」がある. 博士 (工学). 3 次元モデリング, 3 次元曲面データ圧縮, 考古遺物復元などに興味を持つ. 芸術科学会, 映像情報メディア学会, 日本情報考古学会, 情報処理学会, IEEE の会員.