The Visual Computer

Nonlinear ray tracing: visualizing strange worlds

Eduard Gröller

Institute of Computer Graphics,
Vienna University of Technology,
A-1040 Vienna, Karlsplatz 13/186/2, Austria

Nonlinear ray tracing is investigated in this work. Sources of nonlinearity such as gravity centers, gravity lines, chaotic dynamical systems, and parametric curved rays are discussed. Curved rays are represented either iteratively or hierarchically. Algorithms for testing whether a curved ray and a 3D object intersect are presented. Sample images of a test implementation show the feasibility of the approach. Applications of nonlinear ray tracing are the visualization of relativistic effects, visualization of the geometric behavior of nonlinear dynamical systems, visualization of the movement of charged particles in a force field (e.g., electron movement), virtual reality, and arts.

Key words: Ray tracing – Visualization – Nonlinearity – Chaotic dynamical system – Strange attractors

e-mail: groeller@cg.tuwien.ac.at

1 Introduction

Ray tracing is a powerful technique that is widely used in computer graphics for realistic image generation. Light propagation within a 3D environment is simulated by tracing rays through object space to calculate global lighting effects like shading, shadows, reflections, and transparencies (Glassner 1989). Ray tracing determines, for each pixel of the resulting raster image, where the light is coming from. The first object that is intersected by the ray that starts at a given pixel (primary ray) is determined. Secondary rays (reflection rays, transparency rays) are calculated depending on the surface characteristics of the intersected object. Shadow rays (rays between intersection point and light sources) are used to decide whether the point of intersection is in shadow. The evaluation of a shading model produces, in a subsequent step, the color information for the pixel considered. Ray tracing generates realistic images with reasonable computational requirements.
Various extensions and modifications of the basic ray-tracing technique have been investigated to increase efficiency, realism, functionality, etc., (Foley et al. 1990; Glassner 1988; Gröller and Löffelmann 1994). Almost all these approaches make the following assumption on light propagation: light is made up of particles (photons) that travel linearly on straight paths. Although this assumption may hold for a variety of applications, one can think of situations in which linear light propagation is clearly not appropriate. On a macroscopic level, massive objects in space may act as gravity lenses, thereby causing nonlinear light paths (caustics). Relativistic effects due to the local curvature of spacetime require the simulation of curved paths. Imaging of objects that move at a speed close to the speed of light has been investigated by Hsiung et al. (1990) and Ruder et al. (1991).
On a microscopic level it may be better to simulate light as wave forms instead of particles (dual wave-particle nature of light). In Moravec (1981) light is represented by wave fronts to avoid some of the disadvantages (e.g., aliasing) of the particle model in which light propagates in straight and conceptually infinitely thin rays. Applying wave fronts, however, is computationally very expensive. Glassner (1991) briefly discusses some useful applications of curved rays. One example is the visualization of electron particles that due to their interaction with an electric force field, trace out curved paths. Berger et al. (1991) investigate

atmospheric variations (e.g., in temperature or in pressure) that can cause light rays to bend. Mirages are one of the possible visual effects of these bent rays.
In another context, nonlinear rays have already been used to determine the intersection of a straight line and deformed surfaces (Barr 1986) or sweep objects (Kajiya 1983; van Wijk 1984). Instead intersecting a linear ray with a deformed surface, a more cost-efficient test of whether the nonlinearly transformed ray and the undeformed surface intersect is used.
In this paper nonlinear ray tracing is investigated. Examples of applications of nonlinear ray tracing are visualization of relativistic phenomena, visualization of the complexity of nonlinear deterministic dynamical systems and visualization of particle movements (e.g., electrons) that are subjected to a nonlinear force field. Specifying gravity centers with local influence, for example, produces a nonlinear local zoom effect that allows the observer to concentrate on some intersecting portion of object space while viewing the whole object scene. Further applications of nonlinear ray tracing are virtual reality and arts. Not only can one define his own virtual objects and spaces, he can also specify his own virtual (nonlinear) laws of particle or light propagation.

Emphasis has been put on an algorithmic treatment of curved rays. Physical phenomena that produce nonlinear light propagation are modeled only approximately. In Sect. 2 various types of nonlinear rays are presented. Section 3 deals with data structures for efficient ray and object representation. In Sect. 4 ray–object intersection algorithms are discussed. Section 5 discusses a test implementation and resulting images. Section 6 addresses some open problems and further topics on nonlinear ray tracing.

2 Nonlinear rays

In this section various types of nonlinear rays are discussed. Nonlinear rays either are assumed to result implicitly from an underlying dynamic system or are defined explicitly as parametric curves, possibly without a corresponding dynamic model. In the first case, curved rays are assumed to be solutions of a system of first-order ordinary differential equations $\vec{x}' = \mathbf{V}(\vec{x})$ with initial value $\vec{x}_0$. Typically, the exact solution $\vec{x} = \vec{x}_0 + \int \mathbf{V}(\vec{\mathbf{u}})\,d\vec{\mathbf{u}}$ of such a system of differential equations cannot be determined analytically. Therefore, numerical techniques like Euler's method or the more elaborate Runge–Kutta methods are used to calculate approximate solutions to this system of differential equations (Stoer and Bulisch 1983). Section 2.1 discusses nonlinear light propagation in spacetime curved due to gravity centers and gravity lines with either local or global ranges of influence. In Sect. 2.2 nonlinear rays are used to visualize the dynamic behavior and geometric complexity of chaotic dynamic systems. Section 2.3 investigates nonlinear rays that are defined explicitly and analytically through parametric curves.

2.1 Gravity centers and gravity lines

The exact path of a particle influenced by a center or line of gravity is given as the solution of the following second-order differential equation:

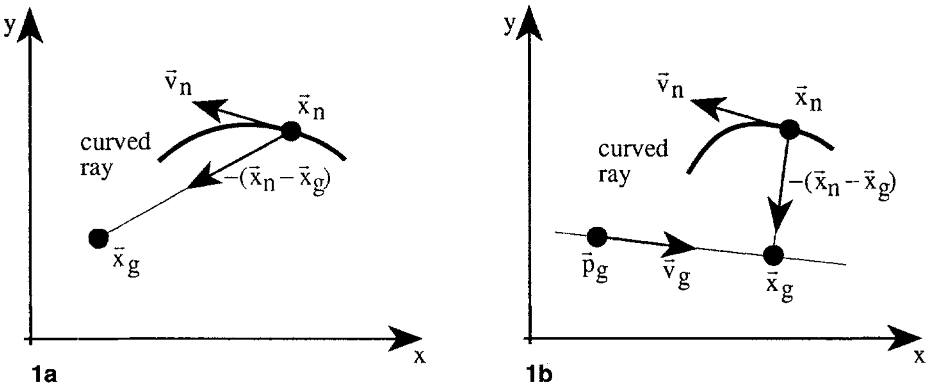
$$\vec{x}'' = -\frac{g \cdot M_g \cdot (\vec{x} - \vec{x}_g)}{\|\vec{x} - \vec{x}_g\|^3}$$

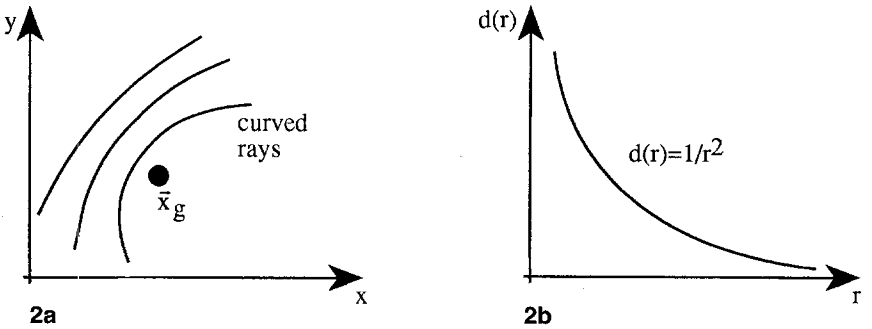
with the initial conditions $\vec{x}(0) = \vec{x}_0$, $\vec{x}'(0) = \vec{\mathbf{v}}_0$. $\vec{\mathbf{x}}(t)$ is the location of the particle in space at time t, $\vec{\mathbf{x}}_0$ is its initial location, $\vec{v}_0$ is its initial speed. $\vec{x}_g$ is the location of the gravity center and M_g its mass. Using Euler's method for solving this second-order differential equation approximately results in the following set of difference equations:

$$\vec{x}_{n+1} = \vec{x}_n + h \cdot \vec{v}_n$$

$$\vec{v}_{n+1} = \vec{v}_n - \frac{g \cdot M_g \cdot (\vec{x}_n - \vec{x}_g)}{\|\vec{x}_n - \vec{x}_g\|} \cdot d(\|\vec{x}_n - \vec{x}_g\|).$$

Rays are assumed to start at a user-defined viewing plane. This determines the initial conditions for our system of difference equations. Parameter h is the step size of the approximating Euler method. The attenuation function $d(\,)$ is introduced to describe the amount and range of influence of a gravity center or gravity line. In the exact physical model the influence of a gravity center or gravity line decreases with the square of the distance. The vector $(\vec{\mathbf{x}}_n - \vec{\mathbf{x}}_g)$ describes the direction





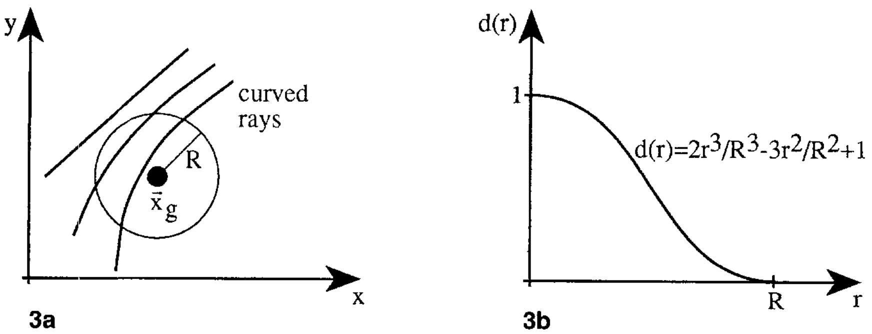


Fig. 1a. 2D gravity center; **b** 2D gravity line

Fig. 2a. 2D gravity center; **b** attenuation function $d(r) = 1/r^2$

Fig. 3a. 2D gravity center; **b** attenuation function $d(r) = 2r^3/R^3 - 3r^2/R^2 + 1$

of influence of a gravity center or gravity line. In case of a gravity center the direction of influence is simply taken to be the vector between the current position $\vec{x}_n$ along the solution (or nonlinear ray) and the point location $\vec{x}_g$ of the gravity center (Fig. 1a). In case of a gravity line the direction of influence is perpendicular to the gravity line. The gravity line is defined by a point $\vec{p}_g$ and a normalized direction vector $\vec{v}_g$. Therefore the direction of influence is given as follows (Fig. 1b).

$$(\vec{x}_n - \vec{x}_g) = \vec{x}_n - \vec{p}_g - ((\vec{x}_n - \vec{p}_g) \cdot \vec{v}_g) \cdot \vec{v}_g.$$

The attenuation function $d(r)$ should be decreasing with increasing distance r to ensure that particles closer to the gravity center or gravity line are influence more than particles farther away. Two attenuation functions $d(r)$ have been investigated: according to Newton's law of gravity $d(r)$ should be taken as $d(r) = 1/r^2$—the influence of a gravity center or gravity line decreases with the square of the distance (Fig. 2).

Although attenuation with $d(r) = 1/r^2$ resembles physical reality, this approach does have two disadvantages in practice. As $d(r) = 1/r^2$ is nonzero for all positive r, a computationally expensive global influence of gravity centers or gravity lines results. Furthermore, particles close to a gravity center or gravity line are greatly influenced due to the singularity of $d(r)$ at $r = 0$.

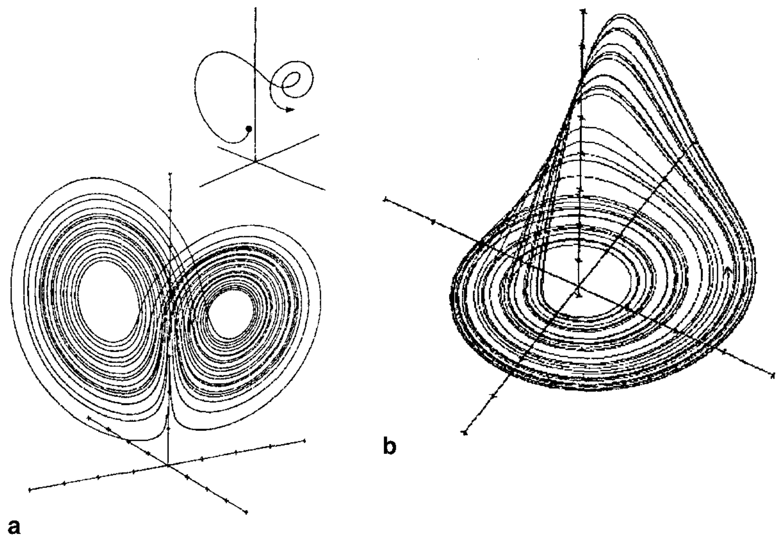


Fig. 4a. Lorenz attractor (Gleick 1988); **b** Rössler attractor (Crutchfield et al. 1989)

In Wyrill et al. (1986a) cubic function that is taken to be nonzero only within a certain range (radius of influence R) is used for modelling soft objects; $d(r)$ is defined as:

$$d(r) = \begin{cases} 2\frac{r^3}{R^3} - 3\frac{r^2}{R^2} + 1 & 0 \leqslant r \leqslant R \\ 0 & \text{else.} \end{cases}$$

The attenuation function $d(r)$ is nonzero only locally and drops smoothly to zero at R (Fig. 3). Gravity centers and gravity lines with local influence allow computationally efficient processing. If several gravity centers and gravity lines are defined, the modified direction of movement of a particle is calculated by summing up the contributions of all gravity centers and gravity lines.

2.2 Chaotic dynamical systems

One way of analyzing nonlinear deterministic dynamic systems is to investigate a suitably defined phase space. A point within this phase space completely determines the state of the system at a certain point in time. The temporal behaviour of such a system results in curved paths, called orbits, within the phase space. The long-term behavior of a nonlinear deterministic dynamic system is often specified by chaotic "strange" attractors with fractal properties (Becker and Dörfler 1989; Gleick 1988; Gröller 1994). Nonlinear ray tracing is useful in visualizing the geometric complexity of such systems. If the paths of primary rays are governed by the differential equations of the nonlinear system, questions as "What do solids existing close to strange attractors look like?" can be easily answered. Thus, conclusions about the local behaviour of dynamic systems (e.g., scaling, stretching, bending effects) can be drawn from the distorted images of solids. Two chaotic dynamic systems are briefly discussed. The Lorenz system is given by the following differential equations (Fig. 4a):

$$x' = \sigma \cdot (y - x)$$

$$y' = r \cdot x - y - x \cdot z$$

$$z' = x \cdot y - b \cdot z$$

with $\sigma = 10$, $b = 8/3$ and $r = 28$. The Rössler system is given by

$$x' = - y - z$$

$$y' = x + a \cdot y$$

$$z' = b + z \cdot (x - c)$$

with $a = 0.375$, $b = 2$ and $c = 4$ (Fig. 4b).

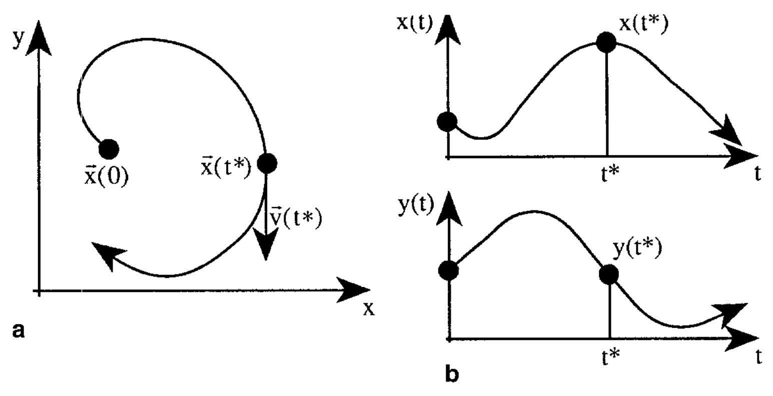


Fig. 5a. A 2D curved ray $\vec{x}(t) = (x(t), y(t))$ defined by parametric functions
b. Parametric functions $x(t)$ and $y(t)$ (scaled by 0.5)

Starting from a given initial position, a solution (orbit, trajectory) of one of these equation sets traces a curved path within phase space that is at any position tangential to the flow defined by the dynamic system. Nonlinear terms in the equations produce highly complex, dynamic behavior. Each trajectory approaches the geometrically complex attractor and follows a highly intricate path on this attracting set. Again a numerical method, such as Euler's can be used to approximate a trajectory of such a dynamic system. An image is generated by specifying an image plane, which means, fixing the starting positions of primary rays (initial conditions). The curved paths of these rays are controlled by the dynamic system itself. Electrostatic fields, magnetic fields, or many other dynamic systems might be taken to govern nonlinear rays.

2.3 Parametric curved rays

Another definition of curved rays can be achieved with parametric functions. The position vector $\vec{x}(t)$ and direction vector $\vec{v}(t)$ of a particle are thereby specified explicitly by parametric functions: $\vec{x}(t) = (x(t), y(t), z(t)$ and $\vec{v}(t) = (x'(t), y'(t), z'(t))$ (Fig. 5).
The scalar coordinate functions $x(t)$, $y(t)$ and $z(t)$ could be derived from an analytically solvable dynamic system or might be specified without any underlying realistic dynamic model in mind. They might be defined as polynomials or trigonometric functions. Curved rays of Sects. 2.1 and 2.2. are approximated incrementally. With parametric curves, however, the entire path of a curved ray is known in advance. This additional information is exploited for efficient ray–object intersection tests. Certain acceleration techniques for the intersection of a curved ray with an object require the determination of the parameter value t for a given position $\vec{x}(t) = (x(t), y(t), z(t))$. Thus, at least one of the coordinate functions $x(t)$, $y(t)$ and $z(t)$ should be invertible. Considering restricted sets of functions only, e.g., lower-order polynomials, enables and simplifies the calculation of minima and maxima, and thus the calculation of a simple bounding box for a curved ray.

3 Data structures

Ray–object intersection tests usually account for most of the cost of the ray-tracing technique. Testing whether an object and a curved ray intersect is even more expensive. One method often employed for tackling nonlinear problems is linearization. This can also be done with nonlinear rays. A curved ray is approximated by a set of linear line segments (Euler integration). A curved ray–object intersection test is transformed into a set of linear ray–object intersection tests. One complicated intersection test is thus replaced by several simple intersection tests. Testing for the intersection of a straight line with many types of objects has been investigated extensively and is reported in the literature (Foley et al. 1990; Glassner 1989; Gröller 1994b). As a curved ray is approximated by line segments all these algorithms can be easily used for nonlinear ray tracing as well. We discuss two representations of curved rays, an iterative representation and a hierarchical

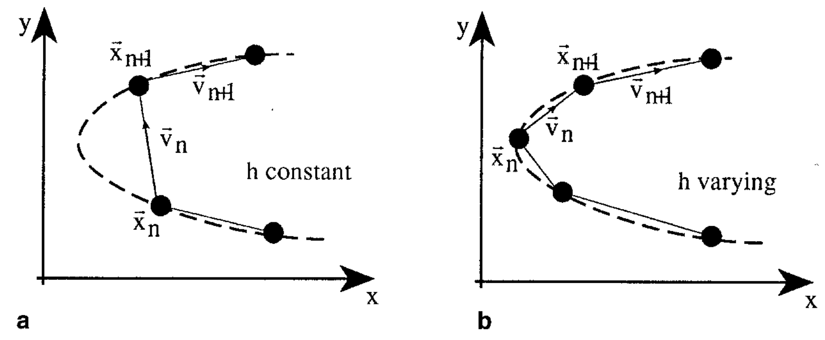


Fig. 6a, b. Iterative ray representation: **a** uniform (h constant); **b** adaptive (h varying)

representation. Although both these representations could be applied to all the types of nonlinear rays discussed previously, we used the iterative representation for rays as defined in Sects. 2.1 and 2.2 and the hierarchical representation for rays as defined in Sect. 2.3. The efficiency of ray–object intersection testing is increased by either object space subdivision or the use of bounding volume hierarchies.

3.1 Iterative ray representation

In this approach a curved ray is represented incrementally by line segments that are given by positions $\vec{x}_n$ and directions $\vec{v}_n$ each. Given a line segment $(\vec{x}_n, \vec{v}_n)$ the following line segment $(\vec{x}_{n+1}, \vec{v}_{n+1})$ is calculated iteratively as follows (see Fig. 6):

$$\vec{x}_{n+1} = \vec{x}_n + h \cdot \frac{\vec{v}_n}{\|\vec{v}_n\|}$$

$$\vec{v}_{n+1} = f(\vec{x}_n, \vec{v}_n).$$

The factor h corresponds to the length of an approximating line segment and can be chosen to be constant or to vary adaptively according to the local curvature of the approximated ray (Fig. 6). Algorithms are simple with a constant h. Varying h adaptively, however, produces better results and greater efficiency.

$\vec{v}_{n+1}$ is calculated by applying the (usually) nonlinear function $f()$. The function $f()$ depends on the type of nonlinearity used (gravity center, gravity line, chaotic dynamical system) and it is derived from formulas like those in Sects. 2.1 and 2.2. Table 1 lists possible choices for function $f()$.

Table 1. Possible choices for function $f()$

Source of nonlinearity	$\vec{v}_{n+1} = f(\vec{x}_n, \vec{v}_n)$
Gravity center, Gravity line	$\vec{v}_{n+1} = \vec{v}_n - \frac{g \cdot M_g \cdot (\vec{x}_n, \vec{v}_n)}{\|\vec{x}_n - \vec{x}_g\|} \cdot d(\|\vec{x}_n - \vec{x}_g\|)$
Lorenz system	$\vec{v}_{n+1} = \begin{pmatrix} \sigma \cdot (y_n - x_n) \\ r \cdot x_n - y_n - x_n \cdot z_n \\ x_n \cdot y_n - b \cdot z_n \end{pmatrix}$ with $\vec{x}_n = (x_n, y_n, z_n)$
Rössler system	$\vec{v}_{n+1} = \begin{pmatrix} -y_n - z_n \\ x_n + a \cdot y_n \\ b + z_n \cdot (x_n - c) \end{pmatrix}$ with $\vec{x}_n = (x_n, y_n, z_n)$
Example of a parametric curve	$\vec{v}_{n+1} = \begin{pmatrix} 2 \cdot x_n \\ 1 \\ 0 \end{pmatrix}$ with $\vec{x}_n = (x_n, y_n, z_n)$

Curved rays leaving the bounding box of the object scene may return at a later point in time or may end up in a loop. In practice, however, only a finite portion of the potentially infinitesimally long curved ray can be traversed. This is done by restricting the ray length, although in certain cases (if the ray length chosen is not large enough) some points of intersection might be missed.

3.2 Hierarchical ray representation

In Ballard (1981) a hierarchical data structure, called a strip tree, for the storage of 2D curves is introduced. Kajiya (1983) uses strip trees to calculate the intersection of a ray and a translational sweep. The concept of strip trees is modified (we use axis-aligned bounding boxes) and extended to three dimensions for the efficient representation of 3D curved rays. A curved 3D ray is stored as a binary tree with bounding boxes (Fig. 7) as follows: the relevant portion of a curved ray is enclosed within an axis-aligned bounding box. Subdividing this bounding box produces two pieces of the curved ray with generally much smaller bounding boxes. These two pieces, together with their bounding boxes, constitute the two sons of the original curve in the hierarchical binary tree representation. A portion of the curve is subdivided recursively until the bounding box is small enough – the enclosed portion of the curve is almost linear or simply monotone. Bounding box calculation and subdivision require the determination of minima and maxima of the curved ray, which is easy if the ray is defined by simple parametric curves such as second-order polynomials. A balanced subdivision is achieved if each node contains a portion of the curved ray of approximately equal length. In the most general case it may happen that an axis-aligned bounding box contains more than one portion of a curved ray. For algorithmic simplicity we avoid such a situation by assuming that every curved ray is monotone with respect to at least one coordinate axis. We subdivide only along axes where monotonicity is given. A bounding box is split in the middle along the axis where monotonicity holds and the bounding box extent is largest. The remaining values of both of the two new axis-aligned bounding boxes are determined by calculating the extremal points of the corresponding coordinate functions.

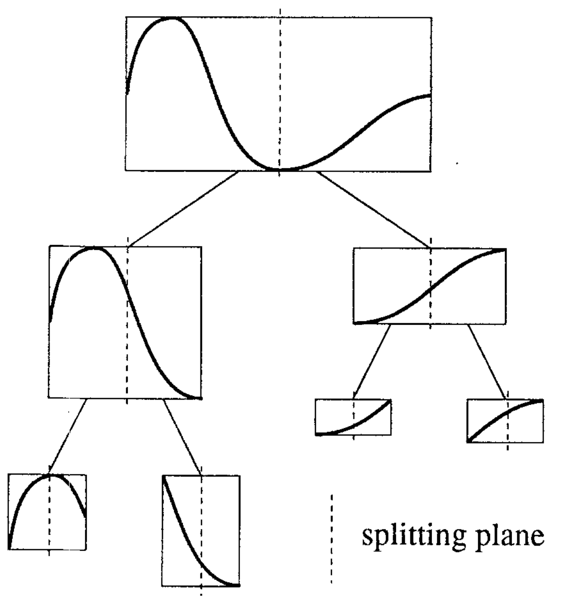


Fig. 7. Hierarchical representation of a curved ray (illustration in 2D) – bounding boxes are split in the middle along an axis where monotonicity of the curve is given

The initial bounding box can be deduced from the smallest bounding box that encloses the objects in object space. Only those portions of a curved ray within this bounding box might intersect objects and are therefore of interest. With the hierarchical storage of a curved ray, large portions of a ray that do not intersect objects can be discarded rapidly by simple bounding–volume comparisons. This data structure is, however, only applicable if a curved ray is known completely in advance. If the curved ray is calculated iteratively, the iterative ray representation of Sect. 3.1 is more appropriate. The hierarchical tree structure can be used to represent sets of adjacent and therefore similar rays as well. With this modification, ray coherence properties can be exploited.

4 Algorithms

The intersection test of a curved ray with an object is replaced by a number of linear ray–object intersection tests. As the linear–object intersection test must be done several times for each

Table 2. Intersection algorithms depending on ray representation and object-space representation

Object \ ray	Iterative	Hierarchical
Subdivision	"sub/iter"	–
Bounding volume hierarchy	"bvh/iter"	"bvh/hier"

curved ray, efficiency is of great concern. Two well-known acceleration techniques, space subdivision (Fujimoto et al. 1986; Glassner 1984) and bounding volume hierarchies (Glassner 1989; Kay and Kajiya 1986 are used for nonlinear ray tracing. Object space subdivision may be uniform (Fujimoto et al. 1986) or non-uniform (Glassner 1984). With bounding volume hierarchies (Kay and Kajiya 1986) objects are grouped together according to their bounding volumes to achieve a hierarchical data structure. A binary bounding volume hierarchy (two bounding volumes are combined to specify the bounding volume of the parent node) has been used for nonlinear ray tracing.
Depending on the ray representation (iterative, hierarchical) and object space representation (uniform subdivision, bounding volume hierarchy) different algorithms result (Table 2). The combination of a hierarchical ray representation and a uniform space subdivision has not been implemented.

4.1 Algorithm "sub/iter"

The intersection test for an iterative ray representation and uniform subdivision is as follows: line segments are processed consecutively until an intersection has been found (if there is any). For each line segment, those subspaces of the subdivision that are intersected by the line segment are determined. This can be done easily by classifying the first and last point of the line segment. If more than one subspace is found, the subspaces are treated in sequential order according to their distance to the starting point of the line segment. Each object that lies at least partially inside the currently processed subspace is tested for linear ray–object intersection. If several intersection points of the line segment with objects of the same subspace exist, the first intersection point is selected.

4.2 Algorithm "bvh/iter"

The intersection test for an iterative ray representation and objects stored in a (binary) hierarchical bounding volume structure is as follows: line segments are again processed consecutively. A line segment is tested to see if it intersects the bounding box of the root node of the bounding volume hierarchy. With axis-aligned bounding boxes, this is efficiently done through comparisons of coordinate values. If the line segment intersects the bounding box of the root node, the two sons of the root are processed recursively. If a leaf node is reached, then the usual linear ray–object intersection test is done. In case of overlapping hierarchies in which the bounding boxes of two nodes at the same level overlap, we must determine if a point of intersection is indeed the first visible one. It does not make sense to determine further points of intersection beyond the first one. In this case if a point of intersection has been found, the search for further points of intersection is, therefore, reduced to a smaller line segment.
Traversal of the bounding volume hierarchy need not restart at the root node for every segment of a curved ray. The information gained during the processing of the previous line segment (with which the current segment even shares one endpoint) is used to focus quickly on the interesting portion of the bounding volume hierarchy.

4.3 Algorithm "bvh/hier"

Objects are organized in a possibly overlapping (binary) hierarchical bounding volume structure. Curved rays are stored in a nonoverlapping binary tree as discussed in Sect. 3.2. In this case, testing whether a curved ray and objects of the scene intersect requires the simultaneous recursive processing of both tree data structures. The data structures and outline of the intersection test algorithm are as follows:

```
ray-node: record
          box:          axis-aligned bounding box;
          left, right:  ↑ray-node;
          end;
```

```
object-node: record
        box:         axis-aligned bounding box;
        left, right: ↑object-node;
        end;
function nonlinear-ray-intersection (r:ray-node, o:object-node):point-of-intersection
var $I_1, I_2$: point-of-intersection;
begin
  if r.box overlaps o.box
  then
    if (o is a leaf node)
    then
      if (r is a leaf node)
      then
        usual linear ray-object intersection;
        return point of intersection if any
      else
        construct r.left, r.right (if not yet existing);
        $I_1$:= nonlinear-ray-intersection (r.left, o);
        If $I_1 < +\infty$
        then
          return ($I_1$)
        else
          $I_2$:= nonlinear-ray-intersection (r.right, o);
          return ($I_2$)
    else
      if (r is a leaf node)
      then
        $I_1$:= nonlinear-ray-intersection (r, o.left);
        $I_2$:= nonlinear-ray-intersection (r, o.right);
        If ($I_1 < I_2$ then return ($I_1$) else return ($I_2$)
      else
        construct r.left, r. right (if not yet existing)
        $I_1$:= nonlinear-ray-intersection (r.left, o.left);
        $I_2$:= nonlinear-ray-intersection (r.left, o.right);
        If ($I_1 < I_2$) then return ($I_1$) else if ($I_2 < I_1$)
        then return ($I_2$);
        If $I_1 < +\infty$
        then return ($I_1$)
        else
          $I_1$:= nonlinear-ray-intersection (r.right, o.left);
          $I_2$:= nonlinear-ray-intersection (r.right, o. right);
          If ($I_1 < I_2$) then return ($I_1$) else return ($I_2$)
  else
    return($+\infty$)/* point of intersection does not exist */
end;
```

The binary tree of a curved ray is constructed on the fly. Portions of this tree are only built when needed. This approach is advantageous, as pieces of the curved ray beyond the first point of intersection are not of interest anyway. *Ray-node* is a node of the binary tree that stores the curved ray and *object node* is a node of the hierarchical bounding volume structure that stores the objects of the scene. Using hierarchical data structures (if possible) results in more complicated algorithms that are, however, more efficient than iterative algorithms.

5 Implementation and results

A test system has been implemented in C++ (Kneidinger 1993). Test runs have been done on a PC 486 although the system is portable to a graphics workstation. Images 1–10 in Fig. 8 were calculated with a resolution of 550×550 and images 11–14, with a resolution of 640×480. Antialiasing was done by oversampling. Table 3 shows some statistics of the sample images. Image 2 shows a regular arrangement of small cubes covering the part of object space we are interested in. Images 3–8 were generated due to the same source of nonlinearity. Taking the regular cube structure of image 2 as object scene clearly illustrates the distortions introduced by the various sources of nonlinearity. Images 3, 5, and 7 show the effect of using curved rays on a more complicated object, whereas images 4, 6, and 8 illustrate the overall effect of these curved rays on object space. Uniform subdivision of object space and the use of bounding volume hierarchies produce algorithms of comparable efficiency.

6 Further topics in nonlinear ray tracing

The test for intersection of curved rays and objects of the scene has been discussed in detail. Ray tracing, however, includes more than just calculating the first point of intersection of primary rays and objects. Shading, shadowing, reflection, and transparency effects can be calculated as well. These tasks become a little bit tricky with nonlinear ray tracing. With linear ray tracing, shadow testing is easy: a linear shadow ray between the point of intersection and the (point) light source is processed. If this shadow ray pierces an opaque object, then the point of intersection is assumed to be in shadow. With nonlinear ray tracing, there is the difficulty of even finding such a shadow ray. Although the endpoints of the shadow ray (point of intersection and position of the light source) are known, due to nonlinear light propagation, it is not obvious from which direction light will reach the point of intersection. This is an inverse problem. Given two points in space, find a curved ray that obeys the underlying laws of nonlinearity and passes through the two defining points. If a

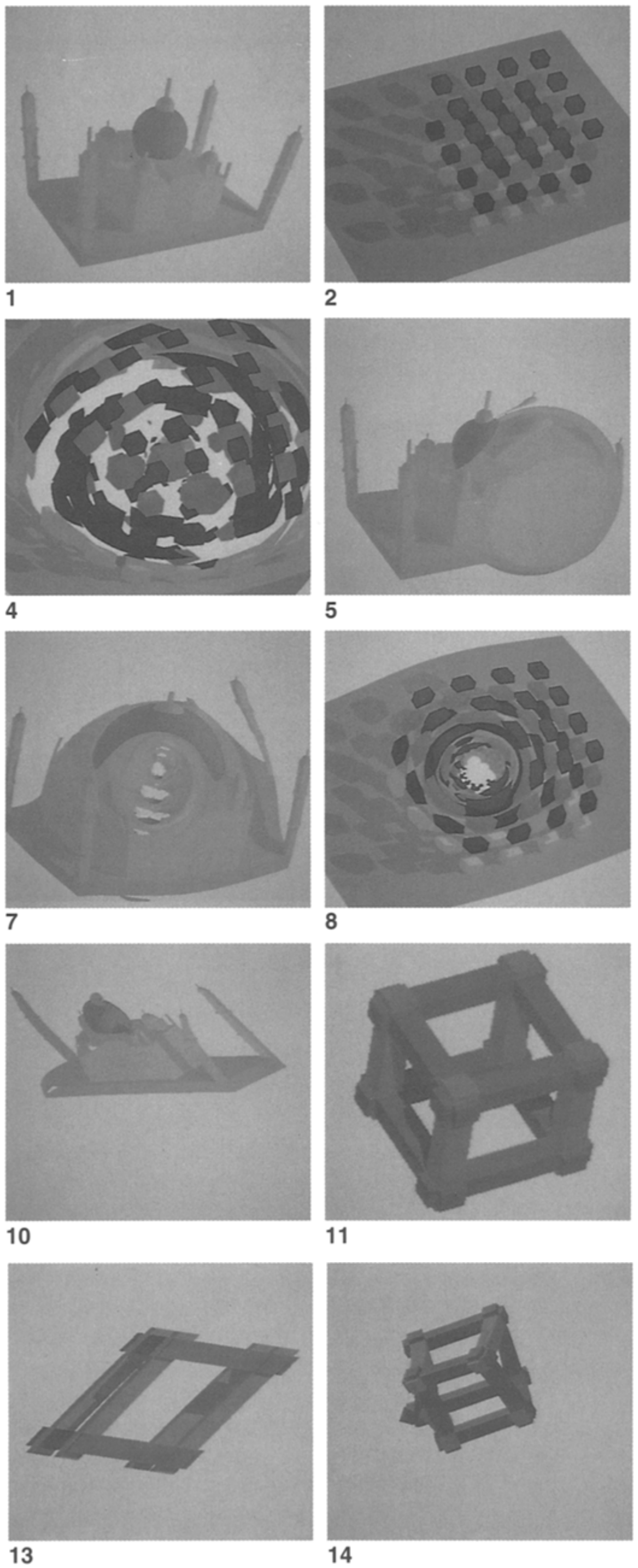


Fig. 8. See Table 3 for a description of images 1–14

Table 3. Description of images 1–14 in Fig. 8

Image number	Algorithm	Remarks
1	linear ray tracing	Reference image; object scene contains 44 objects
2	linear ray tracing	Reference image; object scene contains 65 objects
3	"bvh/iter"	One strong gravity center with global influence. The gravity center is positioned close to the tower farthest in the background
4	"bvh/iter"	Same source of nonlinearity as in image 3, different object scene
5	"sub/iter"	One gravity center with local influence. The gravity center is positioned close to the rightmost tower
6	"sub/iter"	Same source of nonlinearity as in image 5, different object scene
7	"sub/iter"	One gravity line with global influence, the gravity line is perpendicular to the image plane and runs through the center of the image.
8	"sub/iter"	Same source of nonlinearity as in image 7, different object scene
9	"sub/iter"	One gravity line with local influence, The gravity line is parallel to the image plane and runs from the upper left side of the image to the lower right side
10	"sub/iter"	Nonlinearity due to Rössler attractor
11	linear ray tracing	Reference image. Object scene contains 12 objects
12	"bvh/iter"	Nonlinearity due to Lorenz attractor
13	"bvh/hier"	Parametric curve: $x(t) = t^2$, $y(t) = t, z(t) = 2$. Curves are translated so that for $t = 0$ they run through the center of the image
14	"bvh/hier"	Parametric curve: $x(t) = \cos(2t)$, $y(t) = t$, $z(t) = \sin(2t)$. Curves are translated so that for $t = 0$ they run through the center of the image

nonlinear ray results from a system of differential equations, then this problem is a boundary value problem for which methods do exists in applied mathematics. The determination of such a ray seems to be nontrivial but infact, the situation is even worse. One cannot even be assured of the existence of such a ray – a point of intersection might be in shadow although no other object is given to provide shadow. A simple example is a massive gravity center (black hole) close to the light source that prevents light from reaching certain parts of object space. Most shading models take into account the angle of incidence (angle between the normal vector and the incoming light ray). The determination of this angle of incidence poses the same problems for nonlinear ray tracing as for shadow testing. There are various strategies to overcome these limitations to nonlinear ray tracing. One strategy is to assume linear light propagation for shading and shadow calculation. One can think of light particles as being massless (therefore traveling in straight lines as if unaffected by nonlinear forces) until they hit an object. If they strike an object, they pick up some mass and their paths are henceforth influenced by the underlying laws of nonlinearity (e.g., gravity center or gravity line). This model was used in our implementation, although it clearly does not correspond to physical reality. Another strategy is simply to avoid these problems by omitting shading, shadowing, reflection, and transparency calculations. The additional information gained by considering these global lighting effects may not be useful in certain situations. Visualizing the nonlinear behavior of "strange" attractors can produce distorted images of objects that may not be easy to interpret. Taking into account global lighting effects can make the interpretation task even more difficult.

The following idea is a first approach to calculating approximately the correct direction of incoming light at a point of intersection. Object space is subdivided into a regular grid of voxels. Starting at each light source, curved light paths are traced through the voxel structure to give an approximate direction of incoming light for all objects within a given voxel. This structure is generated during a preprocessing step. The angle of incidence is then taken to be the angle between the normal of the point of intersection and the approximate light direction of the voxel that contains the point of intersection.

Aliasing may be more severe with nonlinear ray tracing than with linear ray tracing. All the antialiasing techniques developed for linear ray tracing are applicable to nonlinear ray tracing as well. We have achieved good results with oversampling or adaptive sampling. It must be pointed out, however, that supersampling could fail in some situations. One can think of chaotic dynamic systems with basins of attraction so well mixed in a given domain that any pair of adjacent rays diverge in that domain. In this case, convergence of the sampling process cannot be expected, and sampling must be stopped after an upper limit of sample points has been calculated. The systems we investigated were not too much of a problem in this respect.

Further research topics include reflection and transparency calculations that have not yet been considered. With nonlinear ray tracing the user can also define his own laws of nature and laws of light propagation, such as more than one direction of reflection and/or transparency and antigravity (or repelling gravity) centers.

Acknowledgements The author thanks Gerhard Kneidinger, Werner Komada, and Alexander Zidek who were involved in the implementation of the test system and in generating sample images. Thanks are due to Vojtech Jankovic for fruitful discussions, to Werner Purgathofer for proofreading, and to Wolfgang Meyer for technical assistance. The clarifying remarks of the anonymous reviewers are appreciatively acknowledged.

References

1. Ballard DH (1981) Strip trees: a hierarchical representation for curves. Commun ACM 24:310–321
2. Barr AH (1986) Ray tracing deformed surfaces. Comput Graph 20:287–296
3. Becker KH, Dörfler M (1989) Dynamische Systeme und Fraktale. Vieweg, Braunschweig/Wiesbaden
4. Berger M, Trout T, Levit N (1990) Ray tracing mirages. IEEE Compu Graph Appl 10(3):36–41
5. Crutchfield JP, Farmer JD, Packard NH, Shaw RS (1989) Chaos. In Chaos und Fraktale, Spektrum der Wissenschaft, Heidelberg, pp 8–20
6. Foley JD, van Dam A, Feiner SK, Hughes JF (1990) Computer graphics: principles and practice, 2nd edn, Addison-Wesley, Reading
7. Fujimoto A. Tanaka T, Iwata K (1986) ARTS: Accelerated ray-tracing system. IEEE Comput Graph Appl 6(4):16–26
8. Glassner A (1984) Space subdivision for fast ray tracing. IEEE Comput Graph Appl 4(10):15–22
9. Glassner A (1989) An introduction to ray tracing. Academic Press, San Diego, CA
10. Glassner A (1991) The theory and practice of ray tracing. Eurographics'91, Tutorial No. 1, Vienna,
11. Gleick (1988) Chaos, making a new science. Penguin Books, New York, NY
12. Gröller E (1994) Application of visualization techniques to complex and chaotic dynamical systems. Eurographics Workshop on Visualization in Scientific Computing, Rostock
13. Gröller E (1994) Modeling and rendering of nonlinear iterated function systems. Computers & Graphics 18(5):739–748
14. Gröller E, Löffelmann H (1994) Extended camera specification for image synthesis. Machine Graphics & Vision 3(3):514–530
15. Hsiung PK, Thibadeau RH, Wu M (1990) T-buffer: fast visualization of relativistic effects in spacetime. Comput Graph 24:83–88
16. Kajiya J (1983) New techniques for ray tracing procedurally defined objects. Comput Graph 17:91–102
17. Kay TL, Kajiya J (1986) Ray tracing complex scenes. Comput Graph 20:269–278
18. Kneidinger G (1993) Nichtilineares Ray Tracing, diploma thesis, Technical University of Vienna, Vienna
19. Moravec, HP (1981) 3D graphics and the wave theory. Comput Graph 15:289–296
20. Ruder H, Ertl T, Gruber K, Mosbach F, Subke S, Widmayer K (1991) Kinematics of the special theory of relativity. Eurographics '91, Tutorial No. 12, Vienna
21. Stoer J, Bulirsch R (1983) Introduction to numerical analysis (2nd ed). Springer Berlin Heidelberg New York
22. van Wijk JJ (1984) Ray tracing objects defined by sweeping planar cubic splines. ACM Trans Graph 3:223–237
23. Wyvill G, McPheeters C, Wyvill B (1986) Data structures for soft objects. Visual Comput 2:227–234

EDUARD GRÖLLER is an assistant professor of the Institute of Computer Graphics at the Technical University Vienna. His current research interests include visualization, fractal geometry, nonlinear dynamic systems, and complex rendering techniques. He graduated in Computer Science from the Technical University Vienna in 1987 and did Postgraduate Studies in 1988 and 1989 at the University of Kansas. In 1993 he received his PhD degree from the Technical University Vienna (title of his Ph.D thesis is "Coherence in Computer Graphics"). He is a member of the ACM and the IEEE Computer Society.