Nonlinear Ray Tracing: Visualizing Strange Worlds

Eduard Gröller

Technische Universität Wien
Institut für Computergraphik
A-1040 Wien, Karlsplatz 13/186/2, Austria, EUROPE
Tel.: +43 (1) 58801-4582
FAX: +43 (1) 5874932
e-mail: groeller@cg.tuwien.ac.at

Abstract

In this work nonlinear ray tracing is investigated. Sources of nonlinearity, e.g., gravity centers, gravity lines, chaotic dynamical systems, and parametric curved rays, are discussed. Curved rays are either represented iteratively or hierarchically. Algorithms for the intersection test of a curved ray with a 3D object are presented. Sample images of a test implementation show the feasibility of the approach. Applications of nonlinear ray tracing are 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.

Keywords: ray tracing, visualization, nonlinearity, chaotic dynamical system, strange attractors

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 [Glas89]. 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 which 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 or not. The evaluation of a shading model produces, in a subsequent step, the color information for the pixel in consideration. Ray tracing generates realistic images with reasonable computational requirements.

Various extensions and modifications to the basic ray-tracing technique have been investigated to increase efficiency, realism, functionality, etc. [FoDa90], [Glas89], [GrLö94a]. 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 were 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 in [HsTh90] and [RuEr91].

On a microscopic level it may be better to simulate light as wave forms instead of particles (dual wave-particle nature of light). In [Mora81] light is represented by wave fronts to avoid some of the disadvantages (e.g., aliasing) of the particle model where light propagates in straight and conceptually infinitely thin rays. Applying wave fronts, however, is computationally very expensive. [Glas91] 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. [BeTr90] investigate atmospheric variations (e.g., in temperature or in pressure) which 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 with deformed surfaces [Barr86] or sweep objects [Kaji83], [Wijk84]. Instead of intersecting a linear ray with a deformed surface a more cost efficient intersection test between the nonlinearly transformed ray and the undeformed surface is done.

In this paper nonlinear ray tracing is investigated. Applications of nonlinear ray tracing are for example: visualization of relativistic phenomena, visualization of the complexity of nonlinear deterministic dynamical systems, visualization of particle movements (e.g., electrons) that are subjected to some nonlinear force field. Specifying gravity centers with local influence, for example, produces a nonlinear local zoom effect which allows the observer to concentrate on some interesting portion of object space while having, at the same time, an overall view of the whole object scene. Further applications of nonlinear ray tracing are virtual reality and arts. One may not only define his own virtual objects and spaces but may as well 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 modelled only approximately. In section 2 different types of nonlinear rays are presented. Section 3 deals with data structures for efficient ray and object representation. In section 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 different types of nonlinear rays are discussed. Nonlinear rays are either assumed to result implicitly from an underlying dynamical system or they are defined explicitly as parametric curves, possibly without corresponding dynamical model. In the first case curved rays are assumed to be solutions of a system of first order ordinary differential equations $\vec{x}' = V(\vec{x})$ with initial value $\vec{x}_0$. Typically the exact solution $\vec{x} = \vec{x}_0 + \int V(\vec{u})d\vec{u}$ of such a system of differential equations can not be determined analytically. Therefore numerical techniques like Euler's method or more elaborate Runge-Kutta methods are taken to calculate approximate solutions to the above system of differential equations [StBu83]. Section 2.1 discusses nonlinear light propagation in curved spacetime due to gravity centers and gravity lines with either local or global range of influence. In section 2.2 nonlinear rays are used to visualize the dynamic behavior and geometric complexity of chaotic dynamical systems. Section 2.3 investigates nonlinear rays which 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

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

with the initial conditions $\vec{x}(0) = \vec{x}_0$, $\vec{x}'(0) = \vec{v}_0$. $\vec{x}(t)$is the location of the particle in space at time t, $\vec{x}_0$ 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 approximately solving this second order differential equation results in the following set of difference equations:

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

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

Rays are assumed to start at a user defined viewing plane. This determines the initial conditions for the above 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 $(\vec{x}_n - \vec{x}_g)$ describes the direction 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 (see Figure 1 left). 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 (see Figure 1 right):

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

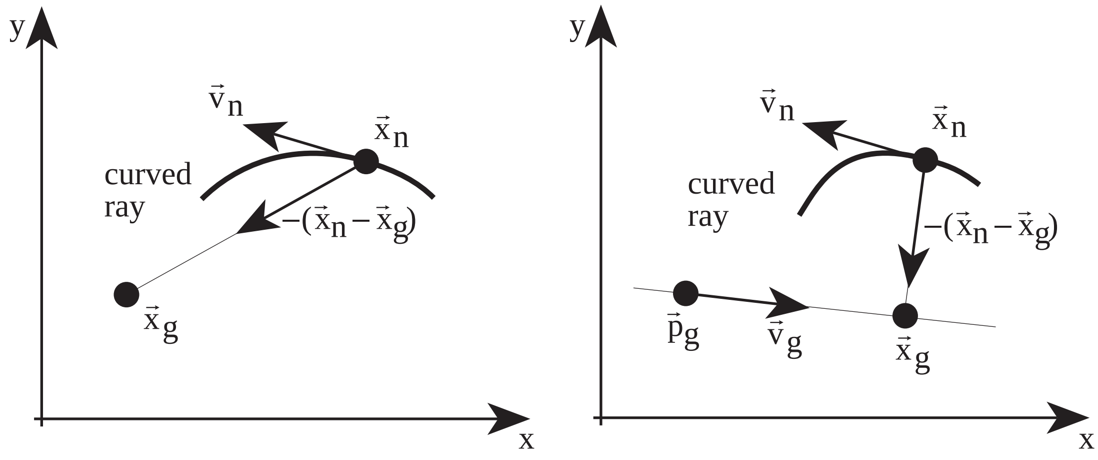

Figure 1: 2D gravity center and 2D gravity line

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 influenced more heavily 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 $d(r)=1/r^2$, i.e., the influence of a gravity center or gravity line is decreasing with the square of the distance (see Figure 2).

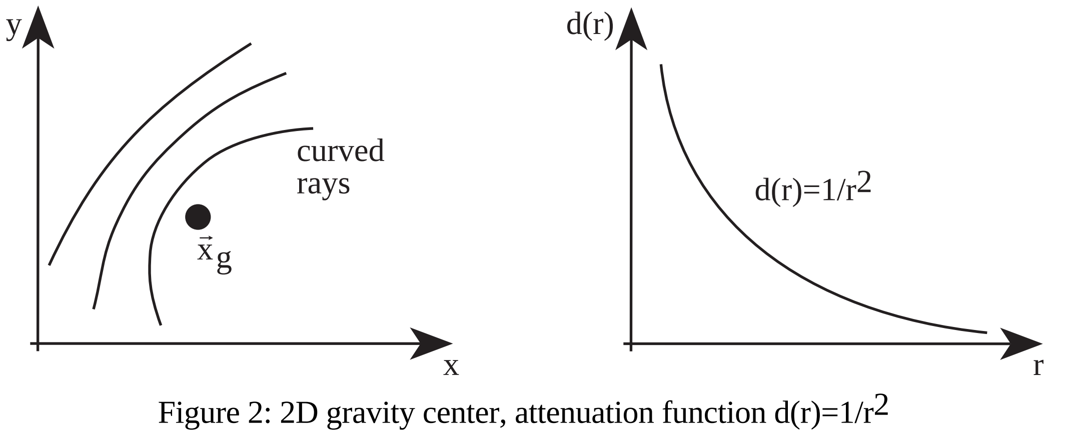

Figure 2: 2D gravity center, attenuation function $d(r)=1/r^2$

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 non zero 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 heavily influenced due to the singularity of d(r) at r=0.

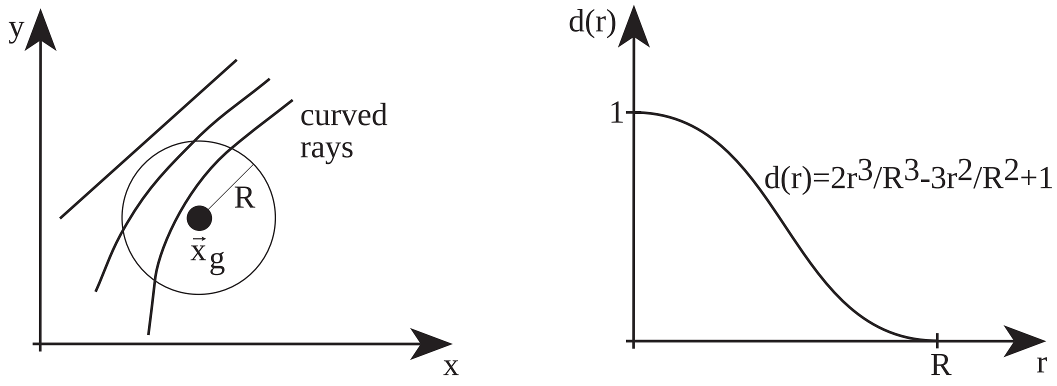

Figure 3: 2D gravity center, attenuation function $d(r)=2r^3/R^3-3r^2/R^2+1$

In [WyMc86] a cubic function that is taken to be non zero only within a certain range (radius of influence R) is used for modeling soft objects. Following [WyMc86] d(r) is defined as:

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

The attenuation function d(r) is non zero only locally and drops smoothly to zero at R (see Figure 3). Gravity centers and gravity lines with local influence allow computationally efficient processing.

If several gravity centers or gravity lines are defined the modified direction of movement of a particle is calculated by summing up the contributions of all gravity centers or gravity lines.

2.2 Chaotic dynamical systems

One way of analyzing nonlinear deterministic dynamical 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 of time. The temporal behavior of such a system results in curved paths, called orbits, within the phase space. The long term behavior of a nonlinear deterministic dynamical system is often specified by chaotic "strange" attractors with fractal properties [BeDö89], [Glei88], [Gröl94a]. 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 living close to strange attractors look like?" are easily answerable. Thus conclusions about the local behavior of dynamical systems (e.g., scaling, stretching, bending effects) can be drawn from the distorted images of solids. Two chaotic dynamical systems will be briefly discussed. The Lorenz system is given by the following differential equations (see Figure 4, left):

$$\begin{aligned} x' &= \sigma\cdot(y-x)\\ y' &= r\cdot x-y-x\cdot z\\ z' &= x\cdot y-b\cdot z \end{aligned}$$

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

$$\begin{aligned} x' &= -y-z\\ y' &= x+a\cdot y\\ z' &= b+z\cdot(x-c) \end{aligned}$$

with a=0.375, b=2 and c=4 (see Figure 4, right).

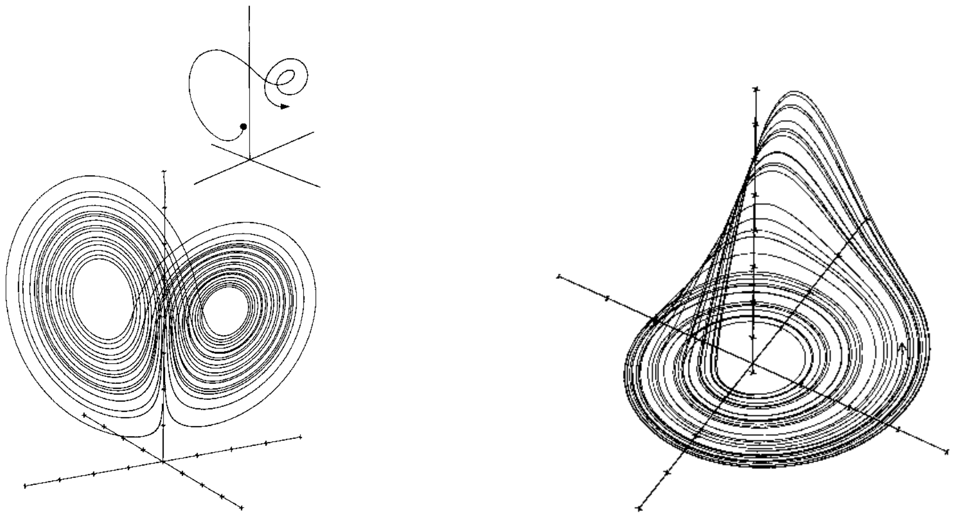
Figure 4: Lorenz attractor [Glei88] Rössler attractor [CrFa89]

Starting from a given initial position a solution (orbit, trajectory) of one of the above equations traces a curved path within phase space which is at any position tangential to the flow defined by the dynamical system. Nonlinear terms in the equations produce highly complex dynamical behavior. Each trajectory is approaching the geometrically complex attractor and follows a highly intricate path on this attracting set. Again a numerical method, e.g., Euler's method may be used to approximate a trajectory of such a dynamical system. An image is generated by specifying an image plane, i.e., fixing the starting positions of primary rays (initial conditions), whereas the curved paths of these rays are controlled by the dynamical system. Many other dynamical systems (e.g., electrostatic or magnetic fields) 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, i.e., $\vec{x}(t) = (x(t),\ y(t),\ z(t))$ and $\vec{v}(t) = (x'(t),\ y'(t),\ z'(t))$ (see Figure 5).

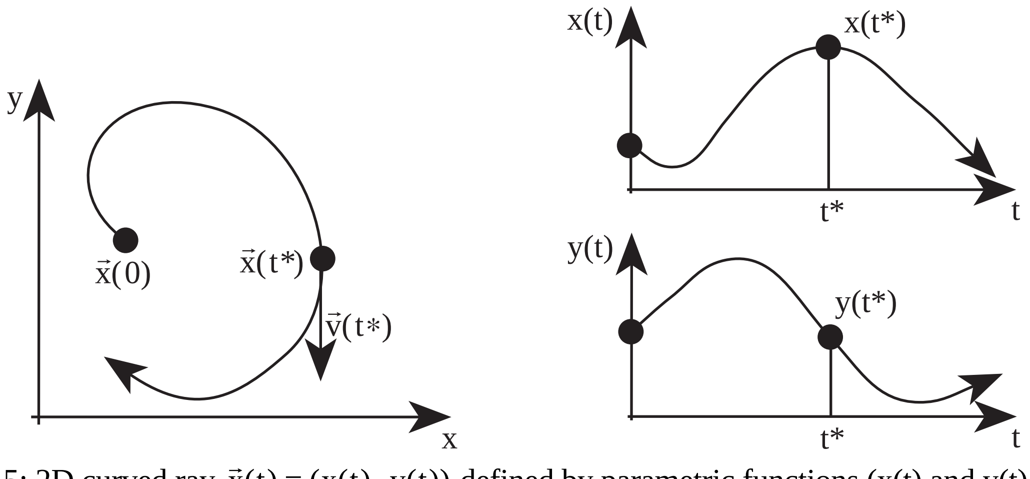

Figure 5: 2D curved ray $\vec{x}(t) = (x(t),\ y(t))$ defined by parametric functions (x(t) and y(t) scaled by 0.5)

The scalar coordinate functions x(t), y(t) and z(t) could be derived from some analytically solvable dynamical system or might be specified without any underlying realistic dynamical model in mind. They might for example be defined as polynomials or trigonometric functions. Curved rays of sections 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 $\vec{x}(t) = (x(t),\ y(t),\ z(t))$. So 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. The intersection test of an object with a curved ray is even more expensive. One often employed method for tackling nonlinear problems is linearization. This can also be done with nonlinear rays. A curved ray is thereby 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. Intersection tests of a straight line with many different types of objects have been investigated extensively in the literature [FoDa90], [Glas89], [Gröl94b]. As a curved ray is approximated by line segments all these algorithms can easily be used for nonlinear ray tracing as well. Two representations of curved rays are discussed in the following: an iterative representation and a hierarchical representation. Although the iterative and the hierarchical representation could both be applied to all types of nonlinear rays discussed previously, we used the iterative representation for rays as defined in sections 2.1, 2.2 and the hierarchical representation for rays as defined in section 2.3. The efficiency of ray - object intersection 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 irections $\vec{v}_n$ 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 Figure 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 may be chosen to be constant or to be varying adaptively according to the local curvature of the approximated ray (Figure 6). With a constant h algorithms are simpler, varying h adaptively, however, produces better results and greater efficiency.

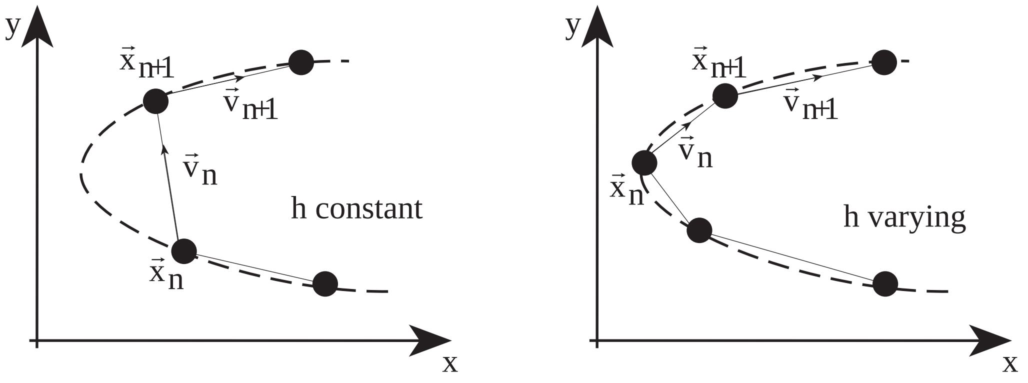

Figure 6: uniform (h constant), adaptive (h varying) iterative ray representation

$\vec{v}_{n+1}$ is calculated by applying the (usually) nonlinear function f(). f() depends on the type of nonlinearity used, e.g., gravity center, gravity line, chaotic dynamical system, and is derived from formulas like those in sections 2.1 and 2.2. Figure 7 lists possible choices for function f().

source of nonlinearity	$\vec{v}_{n+1} = f(\vec{x}_n, \vec{v}_n)$
gravity center, gravity line	$\vec{v}_{n+1} = \vec{v}_n - \frac{g \cdot M_g \cdot (\vec{x}_n - \vec{x}_g)}{\lVert \vec{x}_n - \vec{x}_g \rVert} \cdot d(\lVert \vec{x}_n - \vec{x}_g \rVert)$
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 $\vec{x}_n = (x_n, y_n, z_n)$

Figure 7: possible choices for function f()

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 has not been chosen to be large enough) some points of intersection might be missed.

3.2 Hierarchical ray representation

In [Ball81] a hierarchical data structure called a strip tree for the storage of 2D curves is introduced. [Kaji83] uses strip trees to calculate the intersection of a ray with 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 as follows (see Figure 8): 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. Subdivision of a portion of the curve proceeds recursively until the bounding box is small enough, i.e., 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 can be done easily if the ray is defined by simple parametric curves, e.g., 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. Subdivision is done 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.

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 section 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.

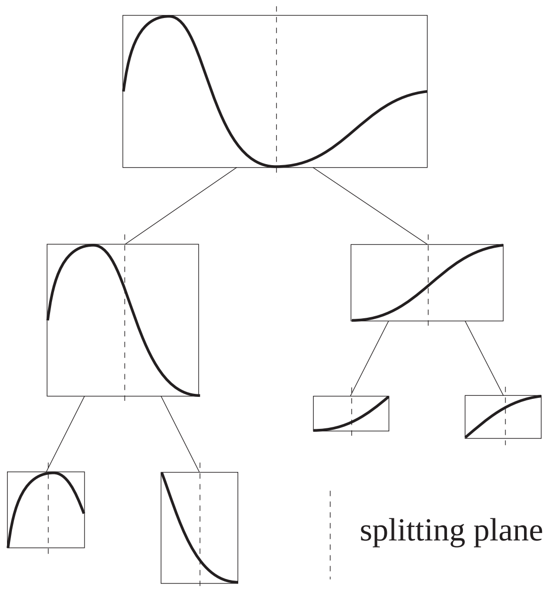

Figure 8: hierarchical representation of a curved ray (illustration in 2D), bounding boxes are split in the middle along an axis where monotonicity of the curve holds

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 ray - object intersection test has to be done several times for each curved ray efficiency is of great concern. Two well-known acceleration techniques, i.e., space subdivision ([FuTa86], [Glas84]) and bounding volume hierarchies ([Glas89], [KaKa86]), are used for nonlinear ray tracing. Object space subdivision may be uniform [FuTa86] or non-uniform [Glas84]. With bounding volume hierarchies [KaKa86] objects are grouped together according to their bounding volumes to achieve a hierarchical data structure. A binary bounding volume hierarchy, i.e., 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 (see Figure 9). The combination of a hierarchical ray representation and a uniform space subdivision has not been implemented.

ray / object	iterative	hierarchical
sub-divison	"sub/iter"	-
bounding volume hierarchy	"bvh/iter"	"bvh/hier"

Figure 9: different intersection algorithms depending on ray representation and object space representation

4.1 Algorithm "sub/iter"

The intersection test for an iterative ray representation and uniform subdivision proceeds 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. A usual linear ray - object intersection test is then performed with each object that lies at least partially inside the currently processed subspace. If there exist several intersection points of the line segment with objects of the same subspace 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 proceeds as follows: line segments are again processed consecutively. A line segment is tested 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, i.e., the bounding boxes of two nodes at the same level overlap, it must be determined if a point of intersection is indeed the first visible one. When a point of intersection has been found it does not make sense to determine further points of intersection beyond the one already determined. In this case the search for further points of intersection is 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 whom the current segment even shares one endpoint) is used to quickly focus 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 non-overlapping binary tree as discussed in section 3.2. In this case the intersection test of a curved ray with objects of the scene requires the simultaneous recursive processing of both tree data structures.

```
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;
```

Figure 10: declarations and algorithm "bvh/hier"

Furthermore 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. In Figure 10 data structures and the intersection algorithm are outlined. *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, for example, iterative algorithms.

5. Implementation and results

A test system has been implemented in C++. Test runs have been done on a PC 486 although the system is easily portable to a graphics workstation. Images 1-10 were calculated with a resolution of 550x550, images 11 - 14 with a resolution of 640x480. Anti-aliasing was done by oversampling. Figure 11 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 and 4, 5 and 6 as well as 7 and 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 different sources of nonlinearity. So images 3, 5 and 7 show the effect of using curved rays on a more complicated object, whereas images 4, 6, 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.

image #	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

Figure 11: statistics of sample images

6. Further topics in nonlinear ray tracing

So far the intersection test of curved rays with objects of the scene has been discussed in detail. Ray tracing, however, includes more than just calculating the first point of intersection between primary rays and objects. Shading, shadowing, reflection and transparency effects may 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 to even come up with 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 some sort of 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 nonlinear ray results from a system of differential equations then the above problem is a boundary value problem for which methods do exist in applied mathematics. The determination of such a ray seems to be nontrivial but the situation is even worse. One cannot even be assured of the existence of such a ray, i.e., a point of intersection might be in shadow although there are no other possibly shadowing objects. 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 shadow testing. There are different strategies to overcome the above mentioned 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 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.

One other strategy is to simply 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 approximately calculate 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 which contains the point of intersection.

Aliasing may be more severe with nonlinear ray tracing than with linear ray tracing. All the anti-aliasing 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 has to be pointed out, however, that there are situations where supersampling could fail. One can think, for example, of chaotic dynamical systems with basins of attraction so well mixed in a given domain that any pair of adjacent rays will diverge in that domain. In that case convergence of the sampling process can not be expected and sampling has to be stopped after an upper limit of sample points has been calculated. The systems we investigated so far were not too much of a problem in that respect.

Further research topics include reflection and transparency calculations which have not been considered up to now. With nonlinear ray tracing the user may also define his own laws of nature and laws of light propagation. Some possibilities are: more than one direction of reflection and/or transparency, anti-gravity centers, i.e., gravity centers that are repelling, etc.

7. References

[Ball81] Ballard, D.H.: Strip Trees: A Hierarchical Representation for Curves, Communications of the ACM, Vol. 24(5), May 1981, pp. 310-321.

[Barr86] Barr, A.H.: Ray Tracing Deformed Surfaces, Computer Graphics, Vol. 20(4), August 1986, pp. 287-296.

[BeDö89] Becker, K.H., Dörfler, M.: Dynamische Systeme und Fraktale, Vieweg publishers, 1989.

[BeTr90] Berger, M., Trout, T., Levit, N.: Ray Tracing Mirages, IEEE Computer Graphics and Applications, May 1990, pp. 36-41.

[CrFa89] Crutchfield, J.P., Farmer, J.D., Packard, N.H., Shaw, R.S.: Chaos, in Chaos und Fraktale, Spektrum der Wissenschaft, 1989.

[FoDa90] Foley, J.D., van Dam, A., Feiner, S.K., Hughes, J.F.: Computer Graphics: Principles and Practice, Addison Wesley, 1990.

[FuTa86] Fujimoto, A., Tanaka, T., Iwata, K.: ARTS: Accelerated Ray-Tracing System, IEEE Computer Graphics and Applications, April 1986, pp. 16-26.

[Glas84] Glassner, A.: Space Subdivision for Fast Ray Tracing, IEEE Computer Graphics and Applications, October 1984, pp. 15-22.

[Glas88] Glassner, A.: Spacetime Ray Tracing for Animation, IEEE Computer Graphics and Applications, March 1988, pp. 103-113.

[Glas89] Glassner, A.: An introduction to ray tracing, Academic Press, 1989.

[Glas91] Glassner, A.: The Theory and Practice of Ray Tracing, Eurographics'91, Tutorial No. 1, Vienna, September 1991.

[Glei88] Gleick, J.: Chaos, Making a New Science, Penguin Books, 1988.

[GrLö94] Gröller, E., Löffelmann, H.: Extended Camera Specification for Image Synthesis, Machine Graphics & Vision, Vol 3(3), 1994.

[Gröl94a] Gröller, E.: Application of Visualization Techniques to Complex and Chaotic Dynamical Systems, Eurographics Workshop on Visualization in Scientific Computing, Rostock, May 1994.

[Gröl94b] Gröller, E.: Modeling and Rendering of Nonlinear Iterated Function Systems, Computers & Graphics, Vol. 18(5), 1994.

[HsTh90] Hsiung, P.K., Thibadeau, R.H., Wu, M.: T-Buffer: Fast Visualization of Relativistic Effects in Spacetime, Computer Graphics, Vol. 24(2), March 1990, pp. 83-88.

[Kaji83] Kajiya, J.T.: New Techniques for Ray Tracing Procedurally Defined Objects, Computer Graphics, Vol. 17(3), July 1983, pp. 91-102.

[KaKa86] Kay, T.L., Kajiya, J.T.: Ray Tracing Complex Scenes, Computer Graphics, Vol. 20(4), August 1986, pp. 269-278.

[Knei93] Kneidinger, G.: Nichtlineares Ray Tracing, diploma thesis, Technical University Vienna, January 1993.

[Mora81] Moravec, H.P.: 3D Graphics and the Wave Theory, Computer Graphics, Vol. 15(3), August 1981, pp. 289-296.

[RuEr91] Ruder, H., Ertl, T., Gruber, K., Mosbach, F., Subke, S., Widmayer, K.: Kinematics of the special theory of relativity, Eurographics'91, Tutorial No. 12, Vienna, September 1991.

[StBu83] Stoer, J., Bulirsch, R.: Introduction to Numerical Analysis (2nd ed.), Springer Verlag, 1983.

[Wijk84] van Wijk, J.J.: Ray Tracing Objects Defined by Sweeping Planar Cubic Splines, ACM Transactions on Graphics, Vol. 3(3), July 1984, pp. 223-237.

[WyMc86] Wyvill, G., McPheeters, C., Wyvill, B.: Data structures for soft objects, The Visual Computer, Vol. 2(4), August 1986, pp. 227-234.

Acknowledgment

The author would like to thank Gerhard Kneidinger, Werner Komada and Alexander Zidek who were involved in the implementation of the test system and in generating sample images.

Thanks 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 thankfully acknowledged.