Changing Some Geometric Parameters in Parameterized Ray Tracing

EDUARDO TOLEDO SANTOS

Escola Politécnica da Universidade de São Paulo
LSI - Laboratório de Sistemas Integráveis
Div. de Sist. Digitais - Grupo de Computação Gráfica
Av. Prof. Luciano Gualberto, travessa 3, n.158
05508-900 - São Paulo, SP, Brasil
`toledo@lsi.usp.br`

Abstract. Parameterized ray tracing is a technique that after the generation of an image by conventional ray tracing, allows changing the *optical* parameters of the scene and rendering a new image in a fraction of the time taken to synthesize the first one. This paper presents an enhancement to this technique that also enables the modification of two *geometric* parameters: light source positions and surface normal vectors (bump mapping).

1. Introduction

After modeling and before rendering, there is an important step in the production of a computer animation or of a single image when one chooses the illumination for the scenery and the colors and textures of its objects. This task requires some aesthetic sense and today is still performed on a trial-and-error basis, demanding some interactivity.

Ray tracing [Whitted (1980)] is the technique of choice when high-realistic images are desired because of its ability in generating reflections, shadows and transparency, including refraction. On the other hand, it is also one of the most time-consuming image synthesis methods, making almost impossible the generation of images at interactive rates.

Introduced in 1989, the parameterized ray tracing technique [Séquin-Smyrl (1989)] allows one to change several optical parameters used for rendering a scene and recompute a new image in much less time than that required for ray tracing the original one. Among the parameters that can be modified are surface colors, diffuse and specular reflection coefficients, transparency and Phong's shine exponent for each object in the scene as well as light source colors and intensities.

Although it speeds up the process of adjusting some optical characteristics of the objects in a scene, parameterized ray tracing does not help in choosing the right place for the light sources, being necessary to fall back on conventional ray tracing when one needs to verify where shadows will be cast if light sources change positions. The same holds true if objects change shape or are simply moved. No geometrical features can be modified in subsequent images.

This paper will present an improved parameterized ray tracing technique that permits the motion of light sources and some changes in the geometrical appearance of objects.

The following sections include a brief overview of ray tracing and parameterized ray tracing, besides the description of the new technique along with suggestions for implementation in both hardware accelerated and software versions.

2. Ray tracing

Ray tracing is a very elegant and simple algorithm for the synthesis of realistic images.

In spite of its simplicity, ray tracing has been used for the generation of the most realistic pictures to date. Even its basic implementation, introduced in 1980 [Whitted (1980)], enables the creation of images featuring shadows, recursive reflection, transparency and refraction. Later enhancements made possible effects like penumbrae [Cook et al. (1984)], motion blur, dull reflections and depth of field [Amanatides (1984)], among others.

The principle of ray tracing is to simulate the optical geometry involved in tracing some of the light rays that travel through the scenery space . For the sake of computational efficiency, ray paths are traced backwards.

Rays originated at the viewer position are cast towards each pixel of the image plane (*primary rays*) See figure 1.

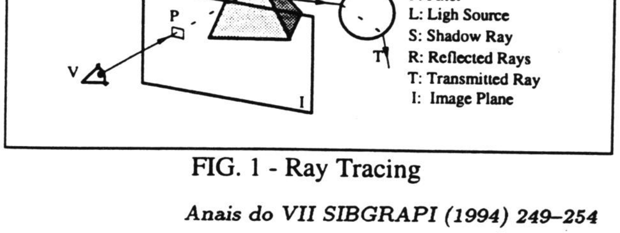

FIG. 1 - Ray Tracing

Then visibility processing takes place: the intersection points of a ray with each object in the scenery are determined and the nearest to its origin is selected. Several techniques have been proposed to accelerate this step [Arvo-Kirk (1989)].

For each visible intersection point, an illumination (or shading) equation must be computed. This equation usually has the formulation below (figure 2):

$$I = k_d . c_{obj} . \sum_{i=\Omega} (\vec{n} \cdot \vec{l}_i) . c_{l_i} + k_s . \sum_{i=\Omega} (\vec{r}_i \cdot \vec{v})^n . c_{l_i} + I_r . k_s + I_t . k_t + I_a$$

where: I	intensity of point on the image plane;
k_d	material diffuse reflection coefficient;
c_{obj}	object color;
Ω	set of light sources 'seen' by the int. point;
$\vec{n}$	unit surface normal vector;
$\vec{l}_i$	unit vector pointing to light source i;
c_{l_i}	light source i intensity;
k_s	material specular reflection coefficient;
$\vec{r}_i$	unit reflection vector;
$\vec{v}$	unit view vector;
n	exponent representing glossiness;
I_r	intensity of reflected ray;
I_t	intensity of transmitted ray;
k_t	material transmission coefficient;
I_a	intensity of global ambient illumination.

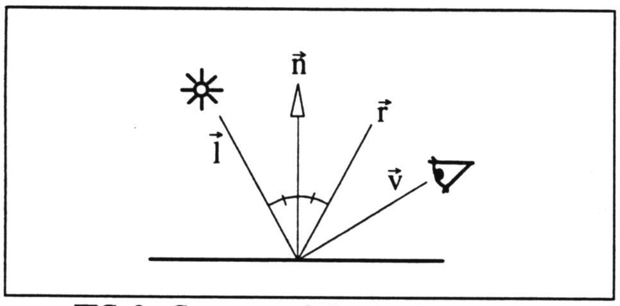

FIG. 2 - Geometry of illumination equation

The first two terms in this equation account for local diffuse and specular illumination while the last three represent the global specular, transmitted and diffuse illumination components.

Evaluation of the illumination equation requires determination of the subset (Ω) of light sources that light each intersection point, i.e., that are directly visible by them. This data can be obtained by tracing the so called *shadow rays* from the intersection points towards the light sources. The detection of any intersection with a shadow ray ends its processing, indicating that there is at least one object that blocks the light from that source.

The global terms I_r and I_t are calculated by casting new rays (called *secondary rays*) in the reflection and refraction directions and by recursively applying the illumination equation until a pre-defined threshold is reached or the ray leaves the scenery (background color).

The set of rays generated for the color computation of a single pixel is called *ray tree*. The ray tree contains the intersection points defined by the ray path through the scenery.

Most of the time spent by the ray tracing algorithm is used for computing ray-object intersections. [Whitted (1980)] reported that, for complex scenes, 95% of the total processing time can be spent in this task (including primary, secondary and shadow rays). Therefore, intersection calculation must be accelerated or eliminated if one wishes to speed up ray tracing.

3. Parameterized ray tracing

Parameterized ray tracing is based on the fact that the ray paths depend only on the geometric characteristics and light sources of a scenery and on the viewer position (with the exception of the index of refraction, an optical characteristic that can change the direction of a ray).

If the object positions, shapes and sizes, as well as light source and viewer positions are held unchanged, then all the costly intersection point and vector calculations necessary for the illumination equation evaluation are kept constant. These calculations are independent of any changes in optical parameters such as surface colors, glossiness, transparency, diffuse and specular reflection coefficients, color and intensities of light sources, color textures, etc. Therefore, one can change optical parameters at will saving most of the processing used for ray tracing and shading equation evaluation.

Changing optical parameters can make the image of a scene very different from the original. To generate figure 4 after the synthesis of the image in figure 3 by ray tracing the following parameters have been changed:

- One more light source was turned on;
- A checker-board pattern was mapped on the table surface;
- The specular reflection coefficient of the teapot was increased.

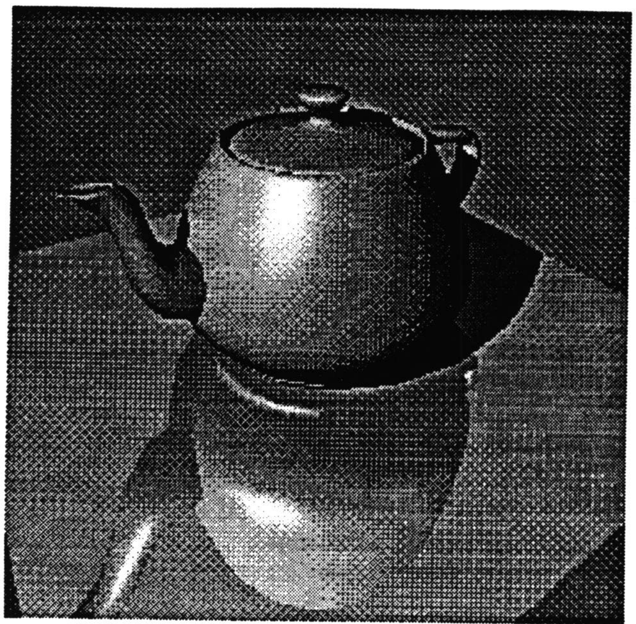
FIG. 3 - Original image, generated by ray tracing

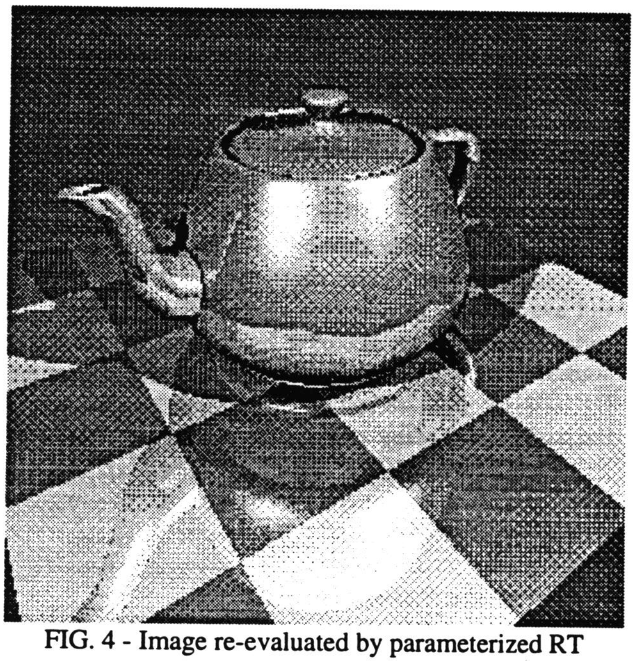
FIG. 4 - Image re-evaluated by parameterized RT

The parameterized ray tracing algorithm stores, for each pixel in an image, information from all the nodes of its corresponding ray tree. Geometrical information is stored as constant values. Optical parameters are indicated by a reference to the corresponding object material.

After a scene has been rendered and a ray tree stored for each pixel, a new image with optical parameters changed can be obtained simply by re-evaluating the equations associated to each pixel.

Because neither intersections nor vector operations have to be recalculated, additional images can be synthesized 50 to 175 times faster than with conventional ray tracing [Séquin-Smyrl (1989)]. This is at the expense of using a rather large amount of file memory (about 10 to 15 times the image file requirements).

Despite its larger use of memory, the parameterized ray tracing technique enables the interactive adjustment of the optical characteristics of a scene, saving a lot of time and effort normally spent in this activity.

4. Light source motion

Using the original parameterized ray tracing technique, one cannot alter the position of the light sources in a scene because this would change the vectors $\vec{l}_i$ and thus, some constant values stored in the ray trees.

Storing the intersection point and surface normal vector, instead of the calculated local illumination terms $(\vec{n} \cdot \vec{l}_i)$ and $(\vec{r}_i \cdot \vec{v})$ for later evaluation, only partially solves the problem because these terms are to be added for i over the subset Ω which can also vary with light source motion. This subset could change as some points, once in shadow, could be exposed to light and vice versa. If not accounted for, these changes can render shadows incorrectly, as they will not move in accordance to the light sources in the scene.

Determining the subset Ω by casting shadow rays should be avoided as this is a very time-consuming operation eliminated in parameterized ray tracing. A better solution should be devised.

We propose the adaptation of two ray tracing acceleration methods that combined can determine which light sources effectively illuminate each point in the ray tree, without fully processing shadow rays. This new technique also preserves one of the advantages of parameterized ray tracing: not requiring access to complete geometric model information in the re-evaluation step.

The mentioned methods are the *light buffer* and the *item buffer* described below.

4.1 The light buffer

The Light Buffer technique [Haines-Greenberg (1986)] accelerates the processing of shadow rays by the use of a direction cube [Arvo-Kirk (1989)] around each light source in a scene.

The direction cube is a data structure that can be represented by a cube centered in the origin of a coordinate system (figure 5). Its faces are orthogonal to the main axes and its edges are 2-unit long, ranging from (-1,-1) to (1,1).

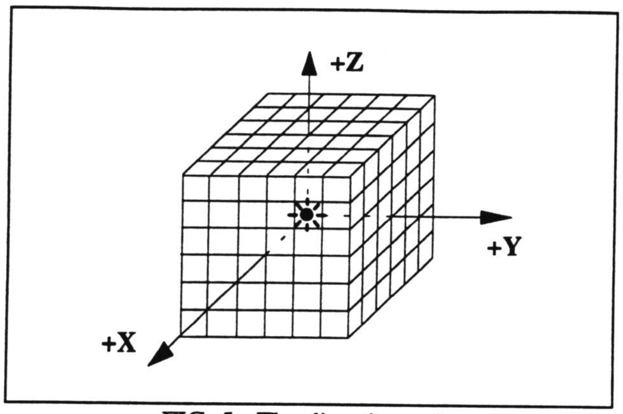

FIG. 5 - The direction cube

Each of the six faces of the direction cube is subdivided in cells that contain information.

When used for implementing a light buffer, the direction cube is centered around a light source and its cells are filled with lists of objects (polygons). These lists indicate which objects are projected (even if partially) onto each cell of the cube, taking the light source as the center of projection. These lists of objects are sorted by ascending distance from the light source. Objects that face away from the light source are not included in the list.

Testing if a point is in shadow is then a simple procedure. For a given light source, the direction of the correspondent shadow ray is calculated. Its intersection with the direction cube is easily determined and also the intercepted cell. Only the objects in the list of this cell are checked for intersection with the shadow ray. If the only intersection found is in the same object (polygon) as the point tested, then the point is not in shadow, in relation to that light source. If the list is checked in the sorted order, the detection of any object further than the point being tested ends the procedure as no other object lasting in the list could block the light of the source to that point.

Several optimizations to this technique are reported in [Haines-Greenberg (1986)].

Note that in the original light buffer algorithm, ray-object intersection calculations are only reduced, but not eliminated. With the new algorithm, this will be achieved.

4.2 The item buffer

The item buffer [Weghorst et al. (1984)] is a technique devised for accelerating primary rays processing. It works exactly in the same way as the z-buffer algorithm but, instead of filling a frame buffer with the color of the visible object at each pixel, a reference to the corresponding object is stored at that location. When processing primary rays, a simple indexing operation can determine the first intersected object .

4.3 A combined approach

The light buffer algorithm is cost effective even though the pre-processing involved in filling out the object lists associated to its cells implies some overhead, specially if there are many light sources in the scenery. Haines and Greenberg reported speed-ups ranging from 2.7 to 12.8.

Most of this overhead is due to the clipping, viewing and perspective transformations. These operations are nowadays commonly performed by dedicated hardware in accelerated graphic workstations.

By combining the light buffer and the item buffer approaches, we can make use of this graphical hardware for speeding up shadow testing [Santos, (1994)], and at the same time do away with the ray-object intersection calculations.

We can think of each face of a direction cube as an item buffer by reducing the cell size to a pixel. Each cell will contain a reference to a single object (the closer one), or none (background).

The hardware associated with the z-buffer usually fills a frame buffer with the color of the object projected over each pixel. Now, we are not interested in the object colors but in a reference to the object. To achieve this goal, each object to be processed by the graphics hardware must be assigned a unique color, its *color-id*, as well as the background. As this kind of hardware often performs color interpolation (Gouraud shading, for example) this feature should be disabled. If this is not available as a rendering option (flat shading), the same result can be obtained by turning off all light sources or making all the reflection coefficients null but the ambient light reflection coefficient, depending on the particular shading equation implemented.

Each face of the direction cubes should be processed as a frame by the hardware. The viewer should be located at the light source position. The image plane corresponds to the directional cube face being processed. Its cells become pixels.

After this processing, each pixel, or cell, will contain the color-id of the object projected over it.

The shadow testing becomes even simpler: if the color-id in the cell pierced by the shadow ray equals that of the object where the point being tested is, than the point is not in shadow in respect to that light source. Otherwise, the point is shadowed.

It is not necessary to calculate intersections anymore due to the fact that, by reducing the cell size to a pixel, only one object (or none) covers the whole cell. Of course this simplification is aliasing prone, but considering the normal use of parameterized ray tracing (adjustment of parameters), some aliasing in the borders of the shadows won't cause much harm. This is specially true because it is common a full regeneration of images or animations by conventional ray tracing after all optical parameters have been set.

As the z-buffer algorithm is very efficient when implemented in hardware, this technique eliminates the overhead associated with light buffer processing. For example, the GTX accelerator for Silicon Graphics workstations is capable of processing 300.000 polygons per second, more than enough for real time calculation of tens of direction cubes, considering usual scenes.

Even if dedicated hardware is not available, a software implementation of the light buffer algorithm or the combined technique are efficient methods for accelerating the determination of the light sources that do not cast shadows on an intersection point.

The figure 6 shows the same scene pictured at figure 4 after one light source has been moved to another position. Note that shadows and highlights as well as global illumination changed accordingly.

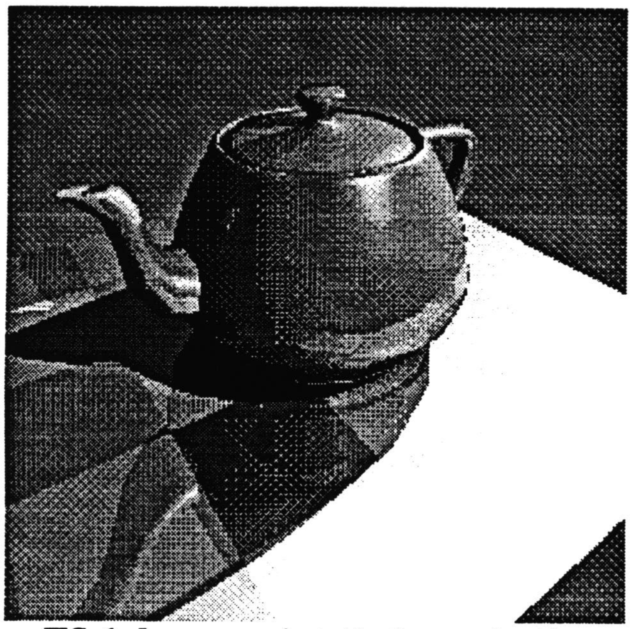
FIG. 6 - Image re-evaluated by the new algorithm

4.4 Drawbacks

Although the combined technique can quickly determine the subset Ω associated to a ray tree node, light source motion also implies the recalculation of both dot products of the shading equation.

These computations introduce an overhead not present in the conventional parameterized ray tracing technique, that must be paid in order to gain the benefits of changing the light source positions, a geometric parameter. We are now investigating the use of graphical hardware to perform this operation, too.

Another drawback that should be reported is related to additional memory requirements. If ω is the number of light sources that light a point (size of the subset Ω), then $2.\omega$ real constants should be stored (8 bytes each) due to the terms $(\vec{n}\cdot\vec{l}_i)$ and $(\vec{r}_i\cdot\vec{v})$. So, 16 ω bytes/node are required in the conventional technique, plus a control register, indicating which light sources are in Ω.

The new technique requires storing the intersection point (3 real numbers) and the normal vector at each ray tree node (3 more real numbers), requiring 48 bytes/node. So, if there are less than 3 light sources in the scenery, more memory will be expended by the new algorithm.

One restriction worth to mention is that the proposed use of a hardware z-buffer requires models composed only by polygonal patches, limiting one of the most important advantages of ray tracing, its capacity of rendering from almost any form of geometric modeling technique.

5. Bump mapping

We can turn one of the drawbacks mentioned earlier in an advantage to our technique.

Because the dot products must be recalculated anyway due to variations in the vectors $\vec{l}_i$ (which also affect the vectors $\vec{r}_i$) we can make a parameter out of the normal vectors too.

This parameter can be used to implement bump mapping [Blinn (1978)], a technique used for generating realistic texture or wrinkles in smooth surfaces. By parameterizing the normal, besides being capable of altering one more geometric parameter, we can save memory, making the requirements reduce to 24 bytes/node.

6. Preliminary results

The images shown in this paper were synthesized with a software called RTp being developed at LSI-USP.

The original image (fig. 3) took 1684 seconds ($\cong$ 28 minutes), at 1000x1000 pixels resolution, in a SGI 4D/480VGX super workstation (using only one processor). The generation of the image in figure 4 took only 4% of this time.

Figure 7 compares the processing times of conventional ray tracing, parameterized ray tracing (for the first image) and image re-evaluation by parameterized ray tracing. Note that the generation of the first image by parameterized ray tracing is slightly slower (7%) than conventional ray tracing because it is necessary to create and store a parameter file.

Figure 8 shows the processing times of the proposed algorithm.

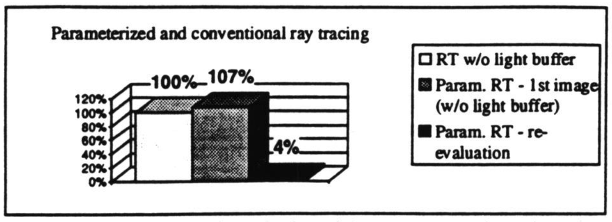

FIG. 7 - Conventional algorithms processing times

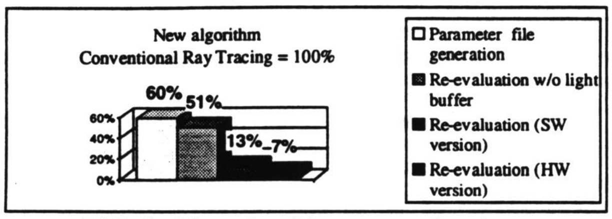

FIG. 8 - New algorithm processing times

When light buffers are used, 13% of the conventional ray tracing time is required to generate the additional images, after the first one. If graphical hardware is employed, this fraction is reduced to about 7%.

7. Conclusions

We presented a new algorithm that enables the modification of geometrical parameters in parameterized ray tracing. With its use, the light sources can be repositioned in a scene without the overhead of a full rendering. Also, bump mapped textures can be modified.

The proposed algorithm takes twice the time of conventional parameterized ray tracing but is still 14 times faster than a full ray tracing, the only other alternative if one needs to move a light source and still keep image quality.

Both surface attribute setting and light source positioning are iterative tasks that are preferably done interactively. Although interactive rates couldn't be achieved yet, efforts are being made towards this goal.

There are several enhancements for reducing the amount of memory used by the algorithm as well as for accelerating its execution time by exploring coherence. We have plans for parallelizing the algorithm and for using graphical hardware for dot product calculation.

The bump mapping parametrization is under implementation in the RTp program.

Acknowledgments

I would like to thank the reviewers for their helpful suggestions and Fernando F. D. Martins for implementing some of the proposed algorithms.

References

T. Whitted, An improved illumination model for shaded display, *Comm. of the ACM* **23** (1980) 343-349.

C. H. Séquin, E. K. Smyrl, Parameterized ray tracing. *Computer Graphics* **23** (1989) 307-314.

R. L. Cook, T. Porter, L. Carpenter, Distributed ray tracing, *Computer Graphics* **18** (1984) 137-145.

J. Amanatides, Ray tracing with cones, *Computer Graphics* **18** (1984) 129-135.

J. Arvo, D. Kirk, A survey of ray tracing acceleration techniques, in: *An introduction to ray tracing* (1989). A. Glassner (ed.). Academic Press, London.

E. A. Haines, D. P. Greenberg, The light buffer: a shadow-testing accelerator, *IEEE Computer Graphics and Applications* **3** (1986) 6-16.

E. T. Santos, Movimentação de fontes de luz em ray tracing parametrizado, *Anais do Congresso Internacional de Computação Gráfica* (1994), CICOMGRAF '94, p.18, SOBRACON, São Paulo.

H. Weghorst, G. Hooper, D. P. Greenberg, Improved computational methods for ray tracing, *ACM Trans. on Graphics* **3** (1984) 52-69.

J. F. Blinn, Simulation of wrinkled surfaces, *Computer Graphics* **12** (1978) 286-292.

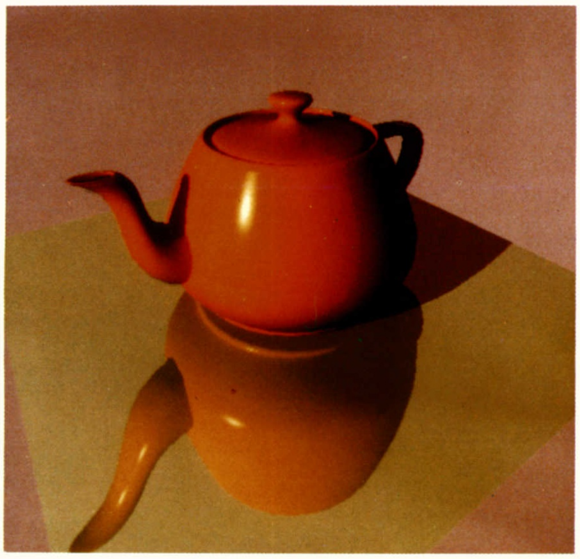
Figure 3

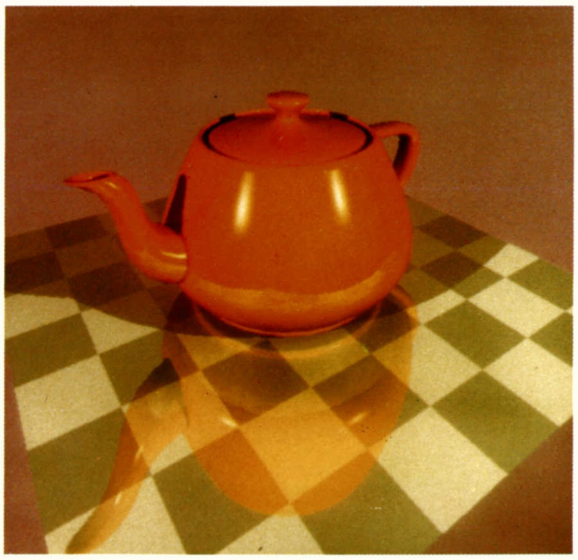
Figure 4

Figuras a cores do artigo ***Changi ıg some geometric parameters in parameterized raytracing***.