A Shading Model for Image-Based Object Rendering

ESTEBAN WALTER GONZALEZ CLUA[1]
MARCELO DREUX[2]
BRUNO FEIJÓ[1]

[1] ICAD, Department of Computer Science, PUC – Rio
{esteban, bruno}@inf.puc-rio.br

[2] ICAD, Department of Mechanical Engineering, PUC – Rio
dreux@mec.puc-rio.br

Abstract. In order to obtain models of complex virtual environments, there are many solutions that make use of image-based rendering techniques. When using a set of images to represent an element in space as a billboard, there are some problems related with the shading calculations, once these elements are just planes in space. This article presents a new approach for shading objects represented by images, which divides the problem into two parts: the shading maps generation for an object with specific set of lights in the environment and real-time shading simulation for a specific observer point of view and object location.

Keywords: Virtual Environments, Textures, Image Based Rendering.

1 Introduction

Virtual environments have many applications, such as games, virtual chat rooms and simulators. A realistic virtual environment typically contains many complex objects and several light sources, which can move independently of each other.

Realistic rendering of each object, with shading, shadows and reflections, is usually too slow for real-time applications. A compromise solution is image-based rendering [Watt et al (2001)]: several reference views of each object are pre-computed, and the proper one is selected, at run-time, based on the relative position of object and camera.

In most of these techniques the scene description is given by a series of reference representations of the scene from arbitrary viewpoints at run-time. In the simplest case of image-based applications there are only information about luminosity and nothing is known about the image depth. Image-based rendering has mostly been studied for systems where the viewer moves through a static environment. However, applications for dynamic scenes have been developed elsewhere [Matusik et al. (2000)].

This article proposes a new illumination model to deal with objects that are represented by images applied to planes. The method is able to handle multiple light sources with surface details approximately shaded.

2 Scene modeling

The image-based modeling of a complete scene can be divided into two parts: the construction of the scene background and the definition of the elements that will be inside the virtual environment.

While the scene background is usually represented by panoramas [Szeliski et al (1997), Matos (1998)], sprites are the best approach to model an object inside the scene. In this work, a *sprite* is a billboard, which is the simplest and most common image-based rendering technique. This is the technique where a texture map in a plane E is considered as a three-dimensional object that can be rotated until it is normal to the view direction towards the camera in the point O. The rotation is applied by an operator $R(E,O)$ whenever the camera or the object moves. This is a property called apparent invariance of rotation. The 2D nature of the plane E is less apparent when the camera is close to the ground and the viewing direction vector is parallel to the ground plane. In this case, the operator R is only applied around the axis perpendicular to the ground plane.

The billboard may have an arbitrary contour. This can be achieved with many techniques. The alpha channel is a very common one and is defined as a byte associated for each pixel that contains the transparency of that part of the image. By making the alpha channel value equivalent

1530-1834/01 $10.00 © 2001 IEEE

to transparent for the pixels of the image that must disappear, its contour may have a shape other than rectangular.

The representation of an object by a plane, how-ever, may produce undesired effects when the camera or the object moves and the image of the sprite remains unchanged. In this case the two-dimensional nature of the object becomes apparent and the 3D illusion deteriorates. Furthermore, the image shading would not be appropriate to the scene lighting.

To solve the above-mentioned problem, the simplest solution is to produce a set of images of the same object, synthesized or photographed from different view angles. According to the camera position in relation to the object, the mapped image has to be appropriately modified. In order to avoid an excessive number of images, the objects and camera are restricted to move parallel to the ground.

This technique is efficient when combined with algorithms that interpolate two object images viewed from different angles. Some authors [Chen et al. (1993)], [Horry et al. (1997)], [Szeliski et al (1997)], [Kanade et al (1997)] propose different approaches to obtain such interpolations. The present paper describes an extension of image morphing, called view morphing [Seitz et al (1996), Seitz (1997)] in order to implement interpolations.

3 A shading model for objects represented by images

When generating an object based on a set of images one of the problems is related to its lighting: the original image shading will rarely correspond to the scene lighting. Even if the shading is adequate to a specific lighting it will not remain as such after an object spatial transformation or viewer change of position.

On the other hand, if there is no shading the scene will be unrealistic, even if textures are applied. Thus, to achieve the shading of an object that matches the existing lighting it is necessary to apply an illumination model to the objects. Nevertheless, the usual lighting models are not adequate to the current situation, where the objects are simple planes with images applied to it. These illumination models would generate a uniform shading over the entire object. Consider, for instance, a column that is being modeled by an image mapped onto a plane, where there is only one light source within the scene. If a conventional illumination model is used the entire column will have uniform shading, although one side of the column is in shadow and the other is being illuminated.

This article proposes a new illumination model to deal with objects that are represented by images applied to planes. The method is able to handle multiple light sources, but surface details are not properly shaded as the surface geometry is not known. To find the right shading it would be necessary to have, in addition to the mapping image, information about the normal vector to each point of the surface. The following algorithm produces real time shading and when combined with the view morphing method it can be a good choice for the modeling of elements inside interactive virtual environments.

3.1 Basic Definitions

The cylindrical shading support, S_c, is defined as a cylinder that completely surrounds a scene sprite, which is simply a plane with an image mapped onto it. The cylinder is invariant to rotation around a vertical axis: if a sprite turns around its center, or the textured image changes, the cylinder will remain static. Thus, if the sprite experiences a translation the cylinder has to be appropriately re-evaluated, once this element will be directly associated with the relative position of the scene lights to its space coordinates.

The borders of the cylinder bases are discretized and divided into equidistant points, which are called components. These components are stored in two lists: L_u e L_l, one for the upper basis and the other for the lower basis, respectively. Each node of these lists is associated to a border component, as shown in figure 1. These lists are defined as sprites shading maps. The origin component is defined as one of the components that touches one side of the sprite plane.

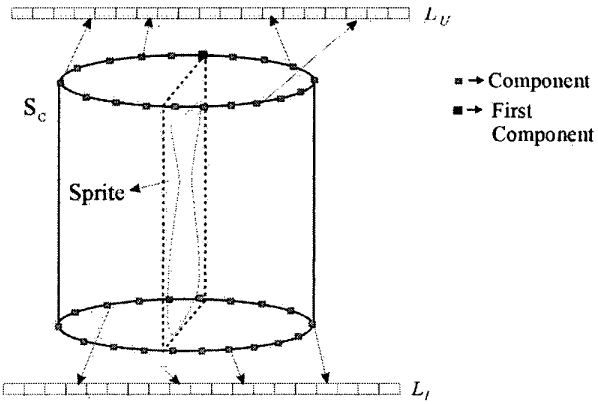

Figure 1 – Cylinder borders are divided into equidistant components. Each one of the upper border components has a node in the list L_u and each one of the lower border components has a node in the list L_l.

Each component $P_k = [X_k\ Y_k\ Z_k]$ from the cylinder border defines two sub-spaces S_{k1} and S_{k2}. These sub-spaces are separate by the plane T_k, that is parallel to Ns, normal vector to the cylinder upper basis, perpendicular to N_k, cylinder vector normal at the point P_k, and passes through the center of the cylinder (figure 2). It is possible to determine where a point Q is located by performing a

dot product between the normalized vector P_k - Q and the plane normal N_k.. If the dot product is positive Q lies on the same sub-space of P_k..

Each node of the L_u and L_l lists is a pointer to a sub-list which contains the illumination vectors of the P_k component associated with the node. The sub-list contains L elements, one for each light source that is present within the same sub-space of the current component.

Illumination vectors are obtained by P_k - I_i, where I_i is the position of the i-th light source. The vector modulus should be equal to the light source intensity.

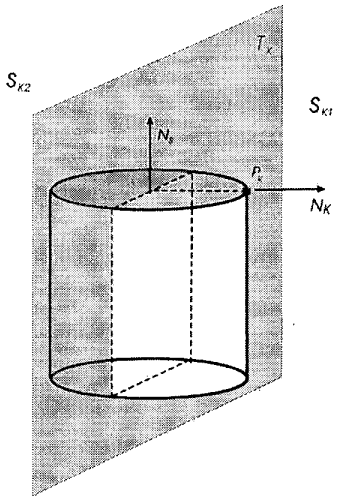

Figure 2 – Each component from the cylinder border defines two sub-spaces S_{k1} e S_{k2}, separate by the plane T_k.

The length of the lists L_u and L_l defines the sprite shading map resolution. The resolution should not be too small to avoid alias problems. A reasonable resolution is given by $2S_{resx}$, where S_{resx} is the horizontal resolution of the texture being mapped. In the case where this shading model is being used combined with the view morphing technique the resolution may not be constant, since the mapping image is being regularly modified. In this situation the shading map resolution could be set to the highest horizontal resolution among all the images being used by the view morphing algorithm.

As the total number of the cylinder border components is $2S_{resx}$, the upper and lower borders are divided in sectors with an angle α given by:

$$\alpha = \frac{\pi}{S_{resx}} \tag{1}$$

3.2 Shading map evaluation

For each light I_i the method traverses all P_k components of the upper and lower borders of Sc, by computing the corresponding lighting map, and adds it to the total lighting map of that sprite in order to build the complete shading map.

The coordinates of each component P_{u_i} and P_{l_i} are calculated by the following equations:

$$P_{u_i} = Sc_u + M_{\alpha_i} V_u$$
$$P_{l_i} = P_{u_i} + Sc_h [0 \quad -1 \quad 0]^T \tag{2}$$

where Sc_u and Sc_l are the centers of the upper and lower cylinder bases respectively, M_{α_i} is the rotation matrix of an angle α_i, V_u is the vector Sc_u - P_{u_0}, and Sc_h is the cylinder height.

Equation 2 applies only if the cylinder basis is parallel to the plane where the camera moves.

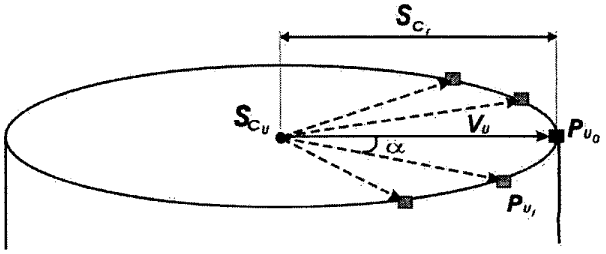

Figure 3 – The cylinder basis are divided in sectors with angle α ($\alpha_i = \alpha$ = constant). P_{u_0} is the coordinate of any point belonging to the upper border and the coordinates of all other components will be obtained from it.

3.3 Sprite shading evaluation

To make use of the proposed method, the original images must be modified before being mapped onto the sprite plane, in order to take advantage of graphics board acceleration facilities. As these facilities do not allow image alteration during the rendering process, it is necessary to make a pre-processing of the sprite image, in order to add shading effects.

There must be a buffer to each object that is being modeled by images. The buffer will store the appropriately modified texture to be applied onto the sprite plane (figure 4). It is important to store the original images in separate files in order to keep the original textures.

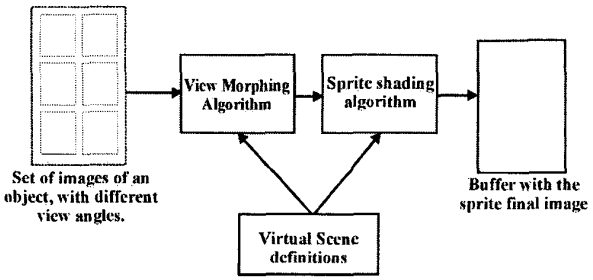

Figure 4 – Schematic representation of the sprite generation pipeline.

Each rendered frame will have a single image projected onto a specific sprite plane. That image is the result of the view morphing of two other images and the application of the proposed shading model. The available buffer should store that image.

The texture shading evaluation process could be divided in three phases:

1)Evaluation of the sprite total rotation angle β;

2)Association of each image pixel column with a node of the shading map;

3)Evaluation to each sprite pixel an illumination vector and application of the proposed shading algorithm to that pixel.

The sprite total rotation angle β is the summation of all rotations applied by R, from the initial sprite position until the current position. To obtain the angle β it is assumed that the observer and the cylinder center have the same vertical coordinate. Thus, the normal to the sprite plane is parallel to the vector $C = O - S_{Cg}$, where O is the observer position and S_{Cg} the cylinder center. This first phase consists in evaluating the angle between vector C and the sprite plane normal, prior to any transformation. The normal should be also stored together with the cylinder data.

The second phase of the shading evaluation is the determination, for each image pixel column, the pair of correspondent nodes into the shading map, one from the upper border and one from the lower border. The pair of nodes is obtained by:

- Determination, for each image pixel, its projection onto the upper and lower cylinder border;
- Determination of the shading map indices, related to the border points.

That evaluation may be performed to an entire pixel column, rather than for individual pixels, since all pixels of a specific column will be projected to the same border points (figure 5). The following algorithm projects each column of a sprite image to a cylinder border, where P_k is the k-th image column coordinate of the border:

For $k = 1$ to C_{resX} resolution of the sprite image do

$$x = \frac{2.k.Sc_r}{C_{resX}}$$

$$y = 0$$

$$z = \sqrt{Sc_r^2 - x^2}$$

$$P_k = M_\beta [x \quad y \quad z]^T$$

In this algorithm, Sc_r is the cylinder radius, C_{resX} is the sprite image x resolution, and M_β is a transformation matrix that will rotate the evaluated point [x y z] of an angle equivalent to the total rotation of the sprite plane applied by the operator R. The cylinder center is supposed to coincide with the center of the global coordinate system.

In the above algorithm, the lower border coordinate is obtained by setting y coordinate to zero The upper border coordinate is obtained by setting the y value to the cylinder height. However, the upper border coordinate is unnecessary, because the correspondent points in the lower and upper border will have the same index in the shading map, as will be shown later.

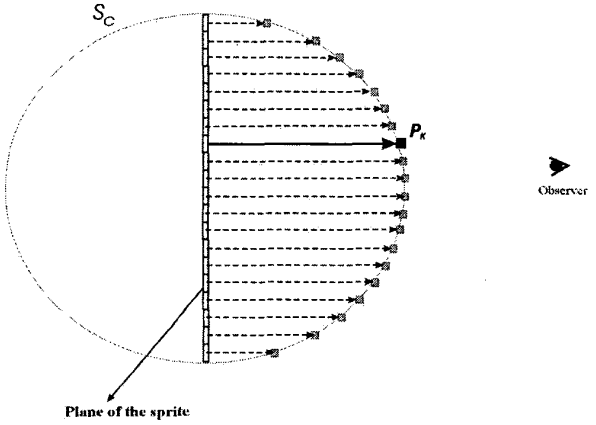

Figure 5 – This image shows the cylinder and sprites from above. The sprite pixels are projected in the direction of the plane normal. The P_k points related to each pixel are the intersection points between the line segment that leaves the pixel in the direction of the normal vector, and points to the observer, with the cylinder border.

If the image resolution remains constant throughout a walkthrough inside the virtual ambient where the object is, only the β angle varies. Hence, it is possible to optimize the algorithm by storing in a table the [x, y, z]

coordinates of each k. By doing that, it is only necessary to perform a search and a rotation operation to each sprite pixel.

If the image resolution remains constant throughout a walkthrough inside the virtual ambient where the object is, only the β angle varies. Hence, it is possible to optimize the algorithm by storing in a table the [x, y, z] coordinates of each k. By doing that, it is only necessary to perform a search and a rotation operation to each sprite pixel.

Once the coordinates of each P_k, that belongs to the cylinder border, are evaluated is time to find out which node of the shading map corresponds to it. As the cylinder has its base parallel to the XZ plane the indices to the upper and lower lists are the same. Thus, to determine the nodes correspondent to the point P_k it is sufficient to choose either the upper or lower base point. This work makes use of the lower border. It is necessary to find the angle θ between the vector $S_{Cl} - P_k$ and the local axis Vy that belongs to the plane where the sprite is being texturized and is perpendicular to its normal. The angle Δ, between the source component and the border point P_k (figure 6), is obtained by the sum of θ and β. That angle is used to index the shading map, in order to find out the light vectors related to the point P_k.

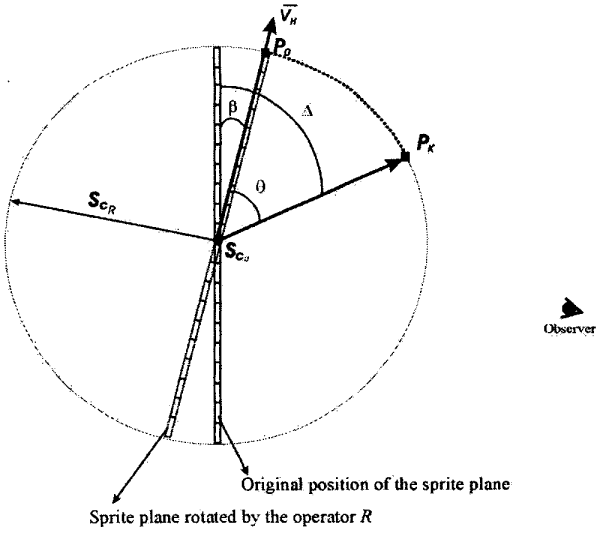

Figure 6 – The angle Δ, obtained by the sum of the angles β and θ, is used to determine the node index of the lists L_u e L_l, related to the cylinder border point P_k The cylinder and the sprite are seen from above.

$$\Delta = \beta + \text{Arccos}\,(\|Vy\| \,.\, \| S_{Cl} - P_k \|) \tag{3}$$

For the evaluation in equation 3, both vectors V_y and $Sc_l - P_k$ must be normalized.

The index of the shading map related to the point P_k is given by the integer part of j in equation 4:

$$j = \frac{2\Delta . C_{resX}}{2\pi} \tag{4}$$

The shading maps store illumination vectors related to the cylinder borders. Once the map index is determined, to a specific pixel column, the algorithm evaluates an approximate illumination vector to each pixel of that column (third phase). A possible approximation is a linear interpolation between the illumination vector from the upper cylinder and the illumination vector of the lower cylinder borders, and considering the distance that the current pixel is from both borders.

Once the shading index k is determined, $L_{u(i)}[k]$ and $L_{l(i)}[k]$ are the illumination vectors of the cylinder extremities for the i-th light source. The illumination vector used to evaluate the shading of the sprite pixel [x, y] is obtained by the following linear interpolation:

$$l_{(x,y)(i)} = (\frac{y}{C_{resY}})L_{u(i)}[k] + (1 - \frac{y}{C_{resY}})L_{d(i)}[k] \tag{5}$$

Where y is the height of the point measured from the cylinder base, in pixels, and C_{resY} is the sprite vertical resolution.

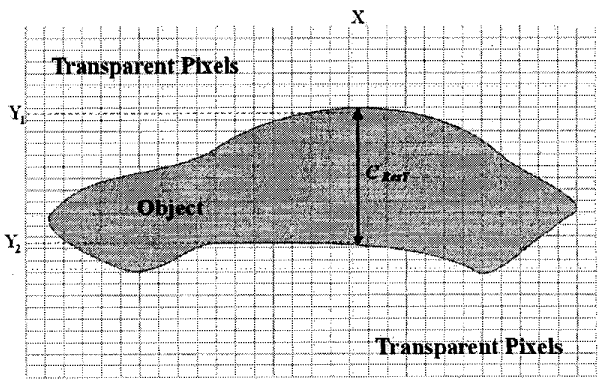

Figure 7 – Each image column has a different C_{resY} value, evaluated as $Y_2 - Y_1$.

However, to achieve a better shading approximation, C_{resY} should be the number of valid image pixels (pixels with transparency index different from zero), rather than the image vertical resolution. Thus, the y index starts

counting from the first line where the pixel is not transparent and ends the counting when it reaches a transparent pixel. To optimize this process, it is recommended to store, within the image data structure, the C_{resY} value of each column and also the position, in pixels, where the y counting should start and finish.

An empiric value to the normal is used when evaluating the illumination to each sprite pixel. It could be obtained as a function of three vectors, as shown in figure 8: the vector N_s, as previously mentioned, is the normal to the upper cylinder basis, $-N_s$ is a vector with its opposite orientation and the vector N_c, normal to the cylinder lateral surface at the point of interest. Thus, the normal vector to each point to be projected is given by N obtained by the following equation and should be normalized afterwards:

$$N = N_c + (1-s)N_s - sN_s$$

And

$$s = \frac{1-y}{C_{resY}} \qquad (6)$$

To obtain a better approximation of the normal vector, some contextual image information should be taken into account. If the sprite images are synthesized, the normal to each point should be stored in addition to its RGB value.

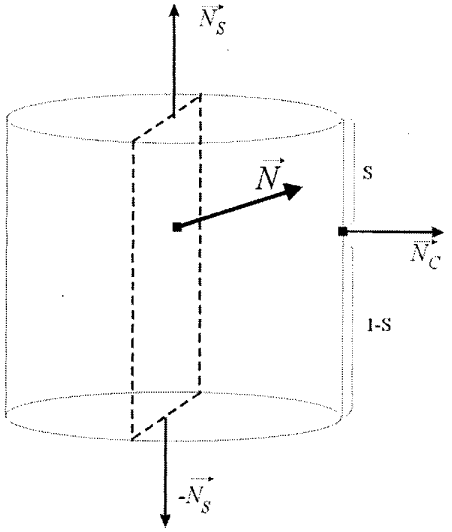

Figure 8 – A proposed solution to obtain the normal is a linear combination of the vectors N_s, $-N_s$ e N_c.

Finally, the shading to be applied to the sprite pixel x,y, with L light sources corresponds to:

$$PixelColor[x,y] = \sum_{i=0}^{L} \left| I_{(x,y)(i)} \right| \left(\left\| I_{(x,y)(i)} \right\| \cdot N \right) PixelColorof\,the\,sprite[x,y] \qquad (7)$$

4 Results and Conclusions

The proposed method has the advantage that it does not consider the object geometry during the shading evaluation. Figure 9 shows a sprite shading without the use of the proposed algorithm. Figures 10 and 11 correspond to sprites shaded with the proposed algorithm.

It can be noticed, as the objects are being represented by images, that the shading is approximate and, depending on the geometry of the objects, the proposed linear interpolation method could be too inaccurate, as it can be shown in Figure 11. Thus, it is convenient to develop and test other interpolation methods and perhaps even to consider pattern recognition in order to obtain some image characteristics.

It should be also allowed to apply spatial transformations to the sprite. As the illumination vectors depend upon the spatial position of the components that are on the cylinder border, it would be necessary to update the vectors every time the cylinder is transformed. This evaluation, however, can be heavily simplified by storing, for each component, its spatial coordinate. In this case when a translation matrix is applied to the sprite and to its cylinder, the transformation would also be applied to each sprite component. Once modified, the illumination vectors should be re-evaluated, which is not an expensive process. For each component only a vector operation to each light source is needed.

5 Acknowledgments

This work was developed in ICAD/PUC-Rio and was partially funded by CNPq.

6 References

[Chen et al. (1993)] Chen, S. E. and Williams, L. View interpolation for image synthesis. Proc. SIGGRAPH 93. *In Computer Graphics* (1993), pp. 279-288.

[Horry et al (1997)] Horry Y., Anjyo K., and Arai K. Tour into the picture: Using a spidery mesh interface to make animation from a single image. In *Proc. SIGGRAPH 97*, pages 225–232, 1997.

[Kanade et al (1997)] Kanade, T., Rander P., and P. J. Narayanan. Virtualized reality: Constructing virtual worlds from real scenes. *IEEE Multimedia*, 4(1):34–46, 1997.

[Matos (1998)] Matos, A. M. Visualização de Panoramas Virtuais. Master Thesis. Pontifícia Universidade Católica do Rio de Janeiro, 1998.

[Matusik et al. (2000)] Matusik, W., Buebler, C., Raskar, R., Gortler, S. J. and McMillan, L. Image-based visual hulls. In Proc. SIGGRAPH 2000, pages 369-374, 2000.

[Szeliski et al (1997)] Szeliski, R. and Heung-Yeung Shum. Creating full view panoramic image mosaics and environment maps. In *Proc. SIGGRAPH 97*, pages 251–258, 1997.

[Seitz et al. (1996)] Seitz, S. M. and Dyer, C. R. View Morphing. Proc. SIGGRAPH 96. *Computer Graphics* (1996), pp. 21-30.

[Seitz (1997)] Seitz, S. M. Image-Based Transformation of Viewpoint and scene appearance. PhD Thesis, University of Wisconsin – Madison, 1997.

[Watt et al (2001)] Watt, Allan and Policarpo, Fabio. 3D games, Real-time Rendering and Software Technology. ACM Press, 2001.

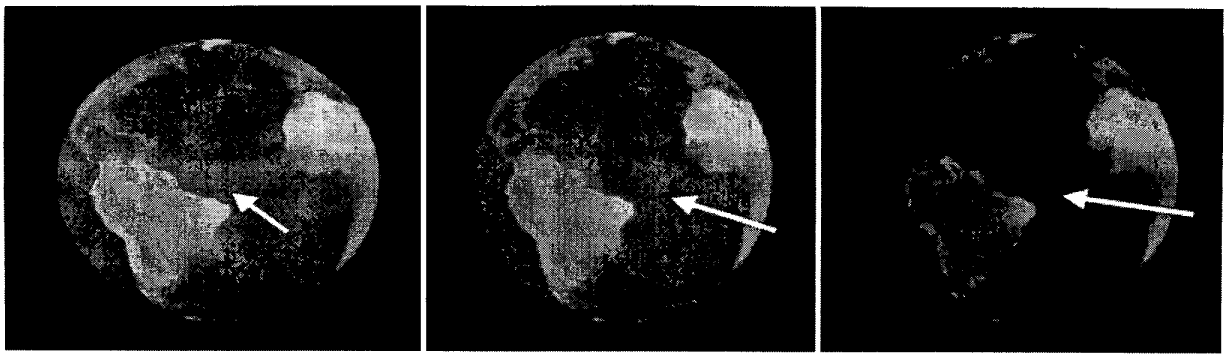

Figure 9 – The image sequence shows a sprite illuminated by a light source located at three different positions. It would be impossible to know where the light source is without the indication of the light source direction, since there is a homogeneous shading.

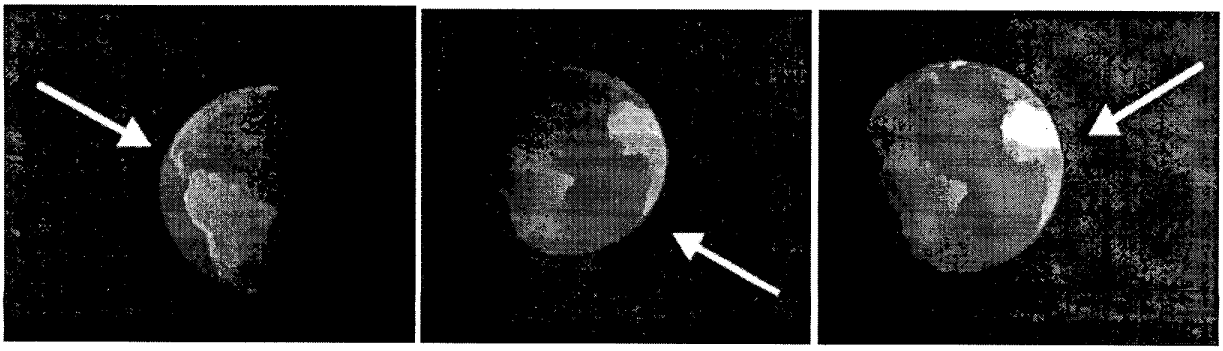

Figure 10 – This figure illustrates the same sprite of figure 9 but with the application of the proposed shading technique. It shows a shading variation which depends on the light source position.

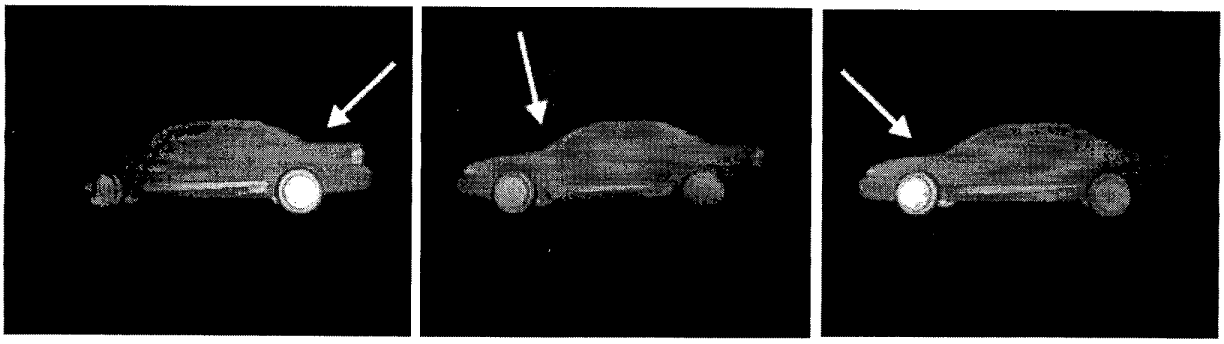

Figure 11 – This sequence shows that the proposed method produces only a shading approximation, since it does not take into account the object geometry.