IMPLEMENTATION OF TWO RELATIVISTIC RENDERERS

Huu Phu Nguyen

Master's Project Report
Under Direction of
Professor Carlo H. Séquin

Computer Science Division
Department of Electrical Engineering and Computer Sciences
University of California
Berkeley, California 94720
Dec 1988

ABSTRACT

Objects passing an observer at a speed that is a significant fraction of the speed of light appear distorted to that observer. There are two contributions to this distortion. The first is the Lorentz contraction which reflects the fact that time and space get interpreted differently in two coordinate systems that move with respect to one another. The second one results from the fact that the photons that bring visual information from the object to the observer have different flying times and thus correspond to different positions of the object in the past. In addition, there are color and brightness changes due to the Doppler effect and due to the differences in photon flux from different parts of the object.

Two renderers implementing these effects have been developed. Their rendering techniques are presented and discussed.

Special Acknowledgements

I would like to give special thanks to my research advisor, Professor Carlo H. Séquin, for his recommendation to this interesting project. His useful suggestions and ideas offered during the development of the relativistic renderers are greatly appreciated.

Also, I would like to thank Professor Manuel Blum for his agreement to read this report on a very short notice.

This work was supported by a gift from Tektronix Inc.

Table of Contents

1. Introduction

When an object passes an observer at relativistic speeds, it appears distorted to that observer. The distortion is due first to the Lorentz contraction in the traveling direction of the object, and second to the different flying times of the photons from the object to the eye of the observer. The Lorentz contraction results from the fact that time and space get interpreted differently in two coordinate systems that move with respect to one another [Fey63,For74,Eis74]. The second source of distortion arises because different photons that bring the visual information of the object to the observer's eyes have different times of flight and therefore correspond to different positions of the object in the past. In addition to the geometrical image distortion, there are color and brightness changes due to the Doppler effect and due to the differences in photon flux from different parts of the object.

Two renderers that implemented these effects have been developed. In order to keep maximum flexibility in choosing the most suitable presentation of results, such as the color mapping between wavelength and hue, the first relativistic renderer has been implemented as a preprocessor. This program maps the geometry of the moving object into a new set of distorted vertex coordinates and assigns new color identifiers to these vertices. This modified scene can then be rendered by standard rendering techniques. One of the Berkeley UniGrafix renderers, UgDisp [Gal86] was enhanced to extend its Gouraud shading capability to the newly introduced vertex color identifiers.

For transparent objects or objects with reflective surfaces, we need an approach more general than the one just described. A second renderer was therefore implemented employing standard ray-tracing mechanisms [Whi80]. In this approach, the vision ray is cast through the scene at the proper speed corresponding to the velocity of light, then its intersection with moving objects is analyzed. This would take care of the distortion due to different flying times of the photons. The Lorentz contraction mentioned above is dealt with in a separate transformation step. The Doppler effect and brightness variations due to changes in photon flux are computed when the vision ray interacts with a moving surface. The relativistic effects on the reflection from or refraction through a moving object are also properly taken into account.

This report is organized as follows: Chapter 2 discusses the mathematics of the distortion effects. Chapter 3 computes the Doppler effect and the brightness change. Chapter 4 presents in detail the implementation of the relativistic renderer preprocessor. Chapter 5 presents the implementation of a Relativistic Renderer and the mathematics behind it. Finally, chapter 6 gives the conclusions and discusses possible future improvements.

2. Distortions

This chapter discusses in detail the two factors of distortion described in the introduction section: the Lorentz Contraction and the distortion due to time-of-flight differences of the photons.

2.1. Lorentz Contraction

According to Special Relativity Theory, time and space get interpreted differently in coordinates systems that travel at different speeds with respect to one another. Suppose that there is a spaceship moving at a speed $\vec{v}$ with respect to some observer A. For convenience, the observer A's coordinates system is chosen in such a way that the velocity vector $\vec{v}$ is in +x direction. Let Δx, Δy, Δz and Δt be the space and time differences between a pair of events measured by A, and $\Delta x'$, $\Delta y'$, $\Delta z'$ and $\Delta t'$ be the space and time differences between the same pair of events but measured by some observer B on the moving spaceship. Then their relationship is given by the Lorentz transformation [For74]:

$$\Delta t = \frac{\Delta t' + (v\Delta x'/c^2)}{\sqrt{1-(v^2/c^2)}} \qquad \{2.1.1\}$$

$$\Delta x = \frac{\Delta x' + v\Delta t'}{\sqrt{1-(v^2/c^2)}} \qquad \{2.1.2\}$$

$$\Delta y = \Delta y' \qquad \{2.1.3\}$$

$$\Delta z = \Delta z' \qquad \{2.1.4\}$$

At any instant in time, the observer A looks at two points on the moving spaceship and notes their space difference as Δx, Δy, Δz, while $\Delta t = 0$. Substituting $\Delta t = 0$ into {2.1.1} and solving for $\Delta t'$, we get

$$\Delta t' = \frac{-v\Delta x'}{c^2}$$

Substituting this equation into {2.1.2}, we then obtain

$$\Delta x = \Delta x'\sqrt{1-\frac{v^2}{c^2}} \qquad \{2.1.5\}$$

From {2.1.3} and {2.1.4} we still obtain

$$\Delta y = \Delta y' \qquad \{2.1.6\}$$

$$\Delta z = \Delta z' \qquad \{2.1.7\}$$

Thus, Δx is a shortened view of $\Delta x'$ by a factor $\sqrt{1-(v^2/c^2)}$, while Δy and Δz are unchanged. It should be noted that $\Delta x'$, $\Delta y'$, $\Delta z'$ are the measurements of the space difference between the two chosen points on the spaceship in the ship coordinates ,i.e., as perceived by observer B on the spaceship. In summary, when an object is moving at speed v with respect to an observer, its dimension in its direction of motion appears shortened by a factor $\sqrt{1-(v^2/c^2)}$ {5.1.8}. This effect is called the Lorentz Contraction. It is simply a non-uniform scaling in the traveling direction with the factor given in {5.1.8}.

2.2. Distortion due To Time-of-Flight Differences of the Photons

Before we can compute the observed distortions of a moving object, we need to define precisely how the visual impressions are recorded. To keep matters simple and unambiguous, we will use a pinhole camera with very short exposure time. In other words, all photons recorded in the image must go through the same point (the pinhole) at the same instant in time. Using such an ideal camera, the distortion can be easily investigated by tracing back the history of these photons as follows [Seq89].

In the past, all photons that have made it through the shutter have occupied spherical shells around the pinhole of the camera. The size of these shells varies with time. The radius of the shell at time t in the past ($t < 0$) is ct, where c is the speed of the photons or the speed of light. In the 3D space coordinate system centered at the pinhole, the shell's equation is

$$x^2 + y^2 + z^2 = (ct)^2 \qquad \{2.2.1\}$$

For convenience, let $w = ct$ (w can be thought of as a renormalized time axis), equation {2.2.1} then becomes

$$x^2 + y^2 + z^2 = w^2 \qquad \{2.2.2\}$$

All of the above photons must have been emitted from some part of the spaceship at a time when that part coincided in space and time with one of these spherical shells of possible photon positions. In particular, we need to look for all points where the time on the shells matches with the time of a particular part position. Only those events can send out photons at the right time to make it through the shutter. Let $\vec{p}_0 = (x_0, y_0, z_0)$ be some point on the spaceship at time $t = 0$. At time t, it is at position

$$\vec{p} = \vec{p}_0 + t\vec{v} \qquad \{2.2.3\}$$

where $\vec{v}$ is the velocity vector of the ship. For convenience, let $\vec{u} = \vec{v}/c$, then equation {2.2.3} becomes

$$\vec{p} = \vec{p}_0 + (ct)\vec{u}$$

or

$$\vec{p} = \vec{p}_0 + w\vec{u} \qquad \{2.2.4\}$$

If $\vec{p} = (x,y,z)$ is the intersection point that we are looking for, it must satisfy both equations {2.2.2} and {2.2.4}, that is

$$|\vec{p}|^2 = w^2 \qquad \text{(from equation 2.2.2)}$$

$$|\vec{p}_0 + w\vec{u}|^2 = w^2 \qquad \text{(from equation 2.2.4)}$$

$$p_0{}^2 + 2w\vec{p}_0 * \vec{u} + w^2u^2 = w^2 \qquad \{2.2.5\}$$

Solving for w, we obtain

$$w = \frac{\vec{p}_0 * \vec{u} - \sqrt{(\vec{p}_0 * \vec{u})^2 + (1-u^2)p_0{}^2}}{1-u^2} \qquad \{2.2.6\}$$

Note that the positive solution of w is discarded because w must lie in the past ($w = ct$ where $c > 0$, $t < 0$).

Substitute w from equation {2.2.6} into equation {2.2.4}, we finally have

$$\vec{p} = \vec{p}_0 + \left[\frac{\vec{p}_0 * \vec{u} - \sqrt{(\vec{p}_0 * \vec{u})^2 + (1-u^2)p_0{}^2}}{1-u^2} \right] \vec{u} \qquad \{2.2.7\}$$

This is the transformation that must be applied to all the points on the ship. The transformed points define the apparent distorted geometry of the spaceship.

3. Brightness and Hue Changes

Two observers moving with respect to one another may perceive different colors on the same object. The hue of the emitted photons is different, first, because their wavelengths are measured differently by the two observers just as time and space get different interpretations, and second, because the photons emitted from a moving object are bunched together in front of the object and spaced out behind it. These effects also change the brightness of an object. Thus, in general, an approaching object appears brighter and bluer while a receding objects appears dimmer and redder.

3.1. Doppler Effect

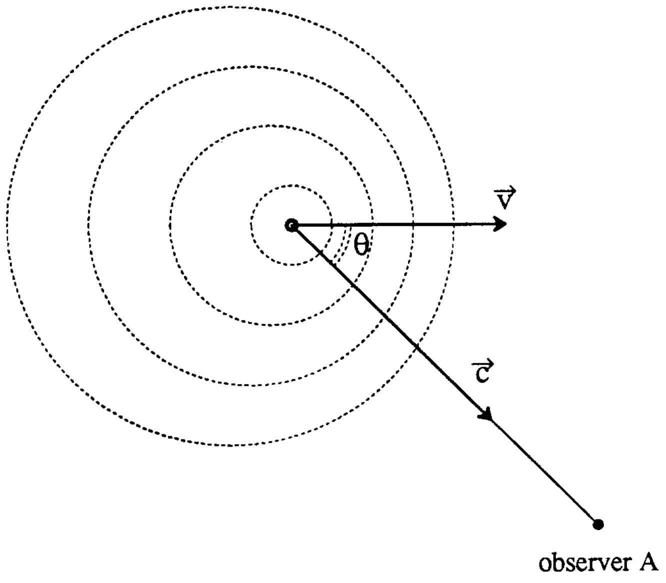


Figure 3.1.1

Figure 3.1.1 shows an object moving at velocity $\vec{v}$ with respect to the observer A. The spacing between the crests of light waves emitted from the moving object, λ', shortened in front of the object by a factor of (c – v)/c, and lengthened behind the object by a factor (c + v)/c. In the direction which makes an angle θ to the velocity $\vec{v}$ (figure 3.1.1), the factor is (c – vcosθ)/c. Thus the wavelength of the light emitted from the object is seen by the observer as

$$\lambda = \lambda' \frac{(c - v\cos\theta)}{c} = \lambda' \left[1 - \frac{v}{c}\cos\theta \right] \qquad \{3.1.1\}$$

where λ' is the wavelength measured in object's coordinates. This is the "classical" Doppler effect where the relativistic effect is ignored. Because the time interval, that is, the period of the wave, is also altered, the time dilation (equation {2.1.1} with $\Delta x' = 0$) must be taken into account. The equation {3.1.1} will then become

$$\lambda = \lambda' \frac{(1 - v \cos\theta/c)}{\sqrt{1 - v^2/c^2}} \qquad \{3.1.2\}$$

Since the frequency f is inversely proportional to the wavelength λ ($f\lambda = c$ = constant in all frames of reference), the equation {3.1.2} can be rewritten as

$$f = f' \frac{\sqrt{1 - v^2/c^2}}{(1 - v \cos\theta/c)} \qquad \{3.1.3\}$$

This formula describes the Doppler Effect on the light emitted by the object. There are three special cases of interest:

Case 1: The object moves directly toward the observer.

Here, $\theta = 0^0$, or $\cos\theta = 1$, equation {3.1.3} becomes

$$f_{ahead} = f' \sqrt{\frac{1 + v/c}{1 - v/c}}$$

The observed frequency will appear higher, or bluer than the actual source frequency. This is called the blue-shift.

Case 2: The object moves directly away from the observer.

Here, $\theta = 180^0$, or $\cos\theta = -1$, equation {3.1.3} becomes

$$f_{behind} = f' \sqrt{\frac{1 - v/c}{1 + v/c}}$$

The observed frequency will appear lower, or redder than the actual source frequency. This is called the red-shift.

Case 3: The observer sees the light at 90^0 or 270^0 to the direction of motion.

Here, $\cos\theta = 0$, equation {3.1.3} becomes

$$f_{perpend} = f' \sqrt{1 - \frac{v^2}{c^2}}$$

This is known as *transverse Doppler effect.* It is a consequence of the Lorentz transformation and has no counterpart for the case of "classical" Doppler effect.

3.2. Photon Flux

Let's assume that an object is moving at velocity $\vec{v}$ with respect to observer A while emitting photons uniformly in all directions.

Due to the object's motion, these photons will be bunched together or spaced out from each other in the same manner as the crests of light waves discussed in the last section behave. When the relativistic Lorentz factor is taken into account, the density of these photons is subject to the same compression or dilation as their frequency [Seq89]. Thus, the brightness perceived by the observer A is

$$I = I' \frac{\sqrt{1 - v^2/c^2}}{(1 - v\cos\theta/c)} \qquad \{3.2.1\}$$

where I′ is the brightness of the photons measured in the moving object's coordinates.

3.3. Hue and Wavelength Mapping

In order to show the Doppler effect, a mapping between the calculated wavelength and the displayed hue is required. In particular, the specified hue of a given surface is converted to a corresponding wavelength where the Doppler effect can be computed, then the resulting wavelength is mapped back to a hue value to be displayed on a color monitor.

Wavelength	(Symbolic Names)	Hue	Comment
780.0 nm	(MAXW)	-60	Upper limit of visible range
700.0 nm	(RED)	0	Standard CIE spectral RED
605.0 nm	(YELLOW)	60	Chosen point for YELLOW
546.1 nm	(GREEN)	120	Standard CIE spectral GREEN
505.0 nm	(CYAN)	180	Chosen point for CYAN
435.8 nm	(BLUE)	240	Standard CIE spectral BLUE
380.0 nm	(MINW)	300	Lower limit of visible range

Table 3.3.1. Control Points for Wavelength and Hue Mapping.

The mapping is made non-linear in order to generate a good looking "rainbow" on the display monitor. We used a uniform cubic Hermite interpolation through the sampling points listed in Table 3.3.1. Then we adjusted the rainbow (displayed on Tek 4129 - it may look different on other monitors) by varying the derivatives at the seven chosen control points.

Hue	ΔHue / ΔWavelength	Comment
-60	1.0*60/(RED-MAXW)	dir. from MAXW to RED
0	1.0*120/(YELLOW-MAXW)	dir. from MAXW to YELLOW
60	0.5*120/(GREEN-RED)	stretch YELLOW by factor 2
120	1.33*120/(CYAN-YELLOW)	reduce GREEN band by factor 0.75
180	0.5*120/(BLUE-GREEN)	stretch CYAN band by factor 2
240	1.0*120/(MINW-CYAN)	dir. from CYAN to MINW
300	1.0*60/(MINW-BLUE)	dir. from BLUE to MINW

Table 3.3.2. Control Slopes for Wavelength and Hue Mapping

The chosen derivative values are given in Table 3.3.2. The resulting mapping is illustrated in Figure 3.3.1.

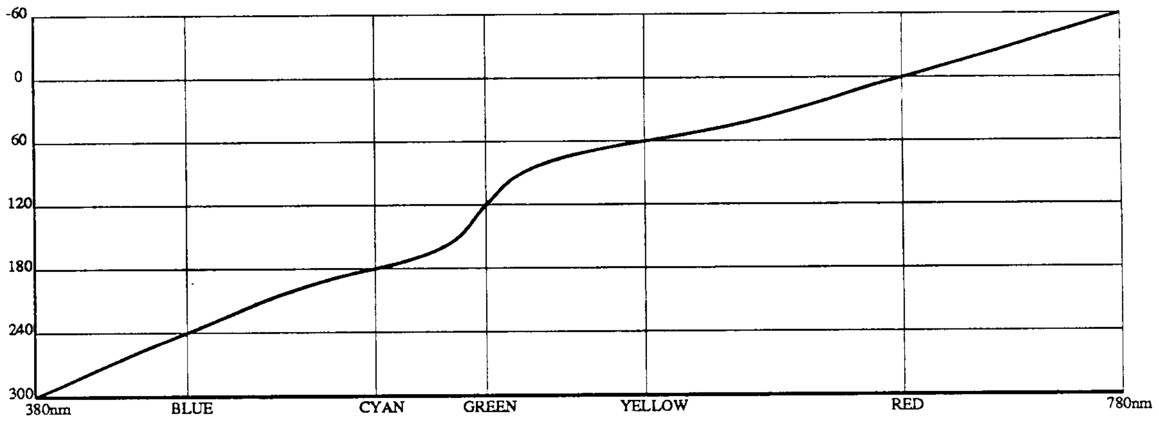


Figure 3.3.1

The human eyes can only "see" electromagnetic waves with wavelengths within a certain range. Our eyes are senseless to "colors" outside this range, and less sensitive to the colors near the boundary of the visible range than those in the middle of the range. To approximate this effect, we chose to attenuate the color intensity according to its wavelength. The attenuation function used in this project is illustrated in figure 3.3.2. The colors in the range [BLUE, RED] will not be affected, colors outside the range [380nm, 480nm] have intensity zero, and for the transition zones (BLUE - 380nm) and (RED - 780nm) we use linearly declining attenuation function.

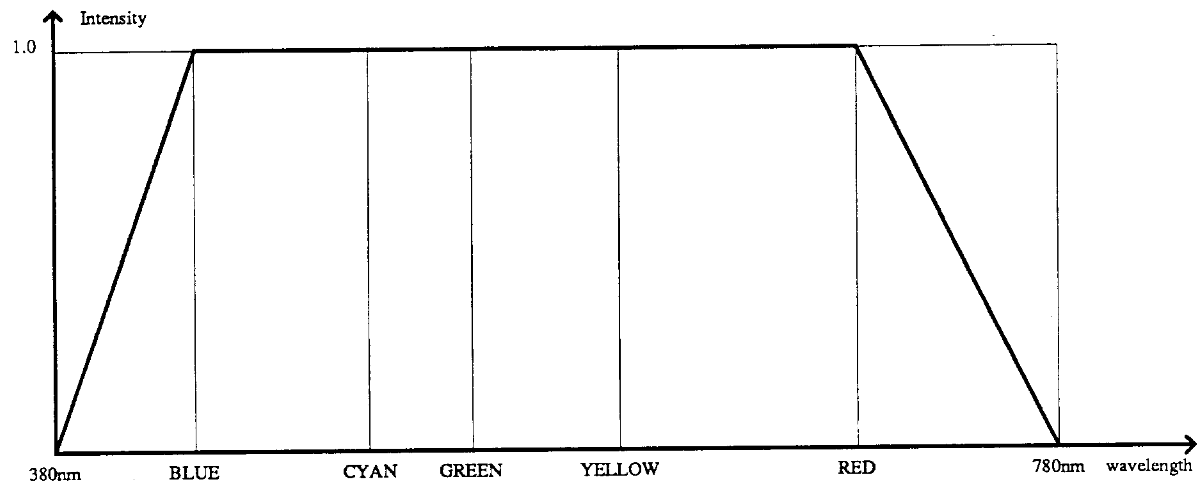


Figure 3.3.2

In the relativistic ray-tracing process (chapter 5) where multiple Doppler effects can occur (from the light source to the moving object, and from the object to the observer), all computation is carried out in the wavelength domain, after initial conversions of the specified hues of all the lights and objects. Only at the very end, when the wavelengths are converted back to hues is the physiologically motivated attenuation applied.

4. Relativistic Renderer Implemented as Preprocessor

The relativistic renderer described in this chapter was implemented as a preprocessor. The preprocessing program maps the geometry of the moving object into a new set of properly distorted vertex coordinates, and assigns proper colors due to Photon Flux and Doppler effects to these vertices. The new scene now can be rendered with standard rendering techniques. One of the Berkeley UniGrafix renderers, UgDisp [Gal86] was modified for this purpose. It is a fast scanline based renderer. Its Gouraud shading capability was extended to linearly interpolate vertex colors in HSV space. The preprocessor and this standard renderer together form a simple relativistic renderer.

4.1. Preprocessors

The preprocessor has to perform two independent tasks: distortion computations and color change computations. Due to their independence, they were designed as two separate modular preprocessors, which were then piped together.

In brief, the first preprocessor computes the distorted vertex coordinates of all vertices while maintaining the topological connectivity of the polygons as originally given. The second preprocessor determines new colors for the modified vertices. There is a potential problem for this scheme. Suppose that a vertex is shared by different polygons having different surface colors. The question is what color should be assigned to this vertex. Emulating a Gouraud vertex shading computation, one would mix the colors of all adjacent polygons in the RGB space. Each R,G, and B component of the vertex color is the average of the adjacent surfaces' corresponding color components. The weighting factor for each face is the angle of the face corner at that vertex. However, this is not a good solution if the user intends to assign different colors to adjacent polygons, yet still wants to have smooth shading across them. For example, a sphere with the top half painted red and the bottom half painted green. The preprocessors developed for this project are therefore enhanced to take an option of specifying this situation. In that case, the vertex is repeated for each face, but in order to retain the smooth shading, a common vertex normal is calculated by averaging the surface normals of all adjacent faces with their angles as the weight factors and assigned to each copy of the vertex (UgDisp was modified to accept these normal vectors).

4.1.1. Distortion Computations

As described in Chapter 2, there are two kinds of distortions: the Lorentz contraction and the distortion due to different flying times of Photons. They will be discussed in turn here.

4.1.1.1. Lorentz Contraction Computation

Recall from section 2.1 that the Lorentz Contraction is simply a non-uniform scaling in the traveling direction with the Lorentz factor $\sqrt{1-(v^2/c^2)}$. Since this scaling applies with respect to the eyepoint (camera pinhole), a translation T is performed if necessary. A rotation transformation R is used to align the velocity vector $\vec{v}$ along the +x direction, before the scaling with the Lorentz factor along the x direction is carried out (no scaling in y or z direction). R could be implemented as two rotations about two major axes; however, there is a more direct way to compute R:

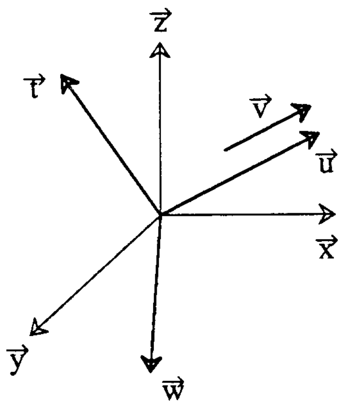


Figure 4.1.1.1.1

Figure 4.1.1.1.1 shows the velocity vector $\vec{v}$ in the original coordinate system xyz. Let's define a new coordinates system uwt so that the positive u direction is in the direction of vector $\vec{v}$. Also let $\vec{u}$, $\vec{w}$, $\vec{t}$ be the **unit basis vectors** of the new coordinate system (figure 4.1.1.1.1), and assume that in the old coordinates, they are $\vec{u} = <u_x, u_y, u_z>$, $\vec{w} = <w_x, w_y, w_z>$, $\vec{t} = <t_x, t_y, t_z>$, then the transformation R is given by

$$R = \begin{bmatrix} u_x & w_x & t_x & 0 \\ u_y & w_y & t_y & 0 \\ u_z & w_z & t_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \{4.1.1.1.1\}$$

The vectors $\vec{u}$, $\vec{w}$, $\vec{t}$ can be computed from $\vec{v}$ and $\vec{x}$, $\vec{y}$, $\vec{z}$ (unit basis vectors for xyz coordinates) from these equations.

$$\vec{u} = \frac{\vec{v}}{|\vec{v}|} \qquad \{4.1.1.1.2\}$$

$$\vec{w} = \frac{\vec{z} \times \vec{u}}{|\vec{z} \times \vec{u}|} \qquad \{4.1.1.1.3\}$$

$$\vec{t} = \frac{\vec{u} \times \vec{w}}{|\vec{u} \times \vec{w}|} \qquad \{4.1.1.1.4\}$$

Care must be taken if $\vec{v}$ and $\vec{z}$ are parallel. In this case, then $\vec{u}$, $\vec{w}$, $\vec{t}$ are redefined to be

$$\vec{u} = \vec{z} \qquad \{4.1.1.1.5\}$$

$$\vec{w} = \vec{x} \qquad \{4.1.1.1.6\}$$

$$\vec{t} = \vec{y} \qquad \{4.1.1.1.7\}$$

Since R is an orthogonal matrix, its inverse is simply its transpose

$$R^{-1} = R^{T}$$

After the affine scaling in x, the inverse matrix R^{-1} (or R^{T}) is applied to undo the rotations. Finally the inverse translation T^{-1} is performed. However, since the next step is the computation of the distortion due to different flying times of photons which also requires the eyepoint at the origin, this inverse translation is postponed until the preprocessor finishes.

4.1.1.2. Computation of the Distortion due to Time-of-Flight

Because the distortion transformation described in section 2.2 maps straight lines into conic sections, the renderer must be able to deal with conic sections directly, or edges must be subdivided enough so that they will be rendered as smooth curves. The renderer used in this project can only take objects consisting of planar polygons as input; therefore, the latter approach must be used. First, the polygons are subdivided by inserting new vertices. Then the transformation given in {2.2.7} is applied to all vertices to create a new scene containing distorted polygons. This scene can now be passed to the preprocessor which computes the color changes. If this preprocessor is implemented independently of the distortion computation, then the inverse translation mentioned in the last section is performed at this time.

4.1.2. Brightness and Hue Changes Computations

This preprocessor is responsible for computing the vertex colors based on the distorted vertex coordinates, their velocities, and their original colors. In other words, it computes the Doppler effect and photon flux effects derived in chapter 2. It then passes all this information to a standard renderer for the actual display.

There are some issues to be discussed here:

Real objects usually reflect lights over a range of wavelengths with varying intensity. To compute the Doppler Effect, all these wavelengths must be appropriately shifted and convolved with the spectrum of the light source, and the result must be shifted again and convolved with the sensitivity function of the observer. To simplify the problem, we assume all surfaces to be of pure spectral (reflecting only a single wavelength). In this particular project, the color must be specified in the HSV color space. Its hue value is then converted to some single wavelength value by the mapping mechanism mentioned in section 3.3. Only this wavelength is considered for the Doppler effect, the saturation value, S, which determines how much white light is contained in the color is left untouched, and the intensity value, V, is adjusted according to the equation of the brightness change {3.2.1}.

Under the Doppler shift, some wavelength may be shifted out of the visible range of human eyes. If this happens then the attenuation function of section 3.3 is used to adjust the color intensity. As the result, the rendered images give us a general idea what people would see.

If the object is moving towards the eyepoint, the resulting brightness for some faces may exceed 1. In such case, the image can be normalized by globally scaling all the intensities (this corresponds to an iris adaptation in the human eye's). If multiple brightness changes may occur (e.g., in relativistic ray-tracing where reflection and refraction are considered), then the normalization or truncation is delayed until the result is ready to be displayed.

The rest of this section describes the operation of the color preprocessor.

Again, a translation that brings the eyepoint to the origin is performed if necessary. In order to minimize the number of color statements output by the preprocessor, we keep track of all colors we generate. A new color statement is output only if the new color does not match with one of the colors we already generated. For each vertex, the following steps are carried out:

(1) The factor $\frac{\sqrt{1-v^2/c^2}}{(1-v\cos\theta/c)}$ of both equations {3.1.2} and {3.2.1} is calculated. To ease the computation, it is rewritten as

$$\frac{\sqrt{1-v^2/c^2}}{(1-\vec{v} * \vec{c}/c^2)}$$

where $\vec{c}$ is the light vector in the direction from the vertex to the eyepoint.

(2) If the vertex color ID matches with one of the colors previously defined and the factor in step (1) is within a certain interval with that color, then we skip

the next three steps.

(3) The original hue value is mapped to its corresponding wavelength which will then be divided by the factor in step (1) to yield the wavelength under the Doppler Effect (section 3.1). The resulting wavelength is finally converted back to the hue value before it is written out.

(4) The original intensity value (V component) is multiplied by the factor in step (1) to yield the new brightness (section 3.2).

(5) Values from steps (3) and (4) along with the original saturation value form a new color for which, a color statement is generated. The color ID for the new color statement is chosen to be the original color ID appended with the factor given in step (1). The new color is recorded for future comparisons.

It is important to note that the attenuation function is not applied here. Such attenuation will be postponed until the color is ready to be displayed by UgDisp.

4.2. UgRainbow - A Modified UgDisp

The renderer chosen for this project was UgDisp [Gal86], a Berkeley UniGrafix renderer. It is scanline based, able to display intersecting objects, and able to perform fast Gouraud smooth shading. It accepts scene descriptions written in the UniGrafix language [SeS83].

In general, the UniGrafix language is a simple language capable of describing 3-D polyhedral objects in boundary representation [Seq83]. Syntactically, a UniGrafix file consists of statements starting with a keyword and ending with a semicolon. There are about a dozen different types of statements, and small additions are made from time to time in a judicious manner. UgDisp, in particular, recognizes eleven of these statements [Gal86]. For our purposes, the syntaxes and meanings of all statements were kept the same, except for the vertex statement. This statement which is used to define a vertex ID and vertex coordinates was extended to accept a color ID and an optional vertex normal specification (see section 4.1). The enhanced UniGrafix language is employed by the preprocessors to pass all necessary information to the modified UgDisp.

The modifications made to the UgDisp were moderate. First, the parser must be able to accept the extended vertex statement. Second, the existing data structures must be modified to contain extra information for each vertex, i.e. its color and vertex normal. Finally, the Gouraud shading was enhanced to linearly interpolate vertex colors in HSV space. Only this change was significant enough to be discussed here in more detail.

The color statement in the UniGrafix language specifies a color by its HSV values. In UgDisp, the Gouraud shading module only interpolates the illumination value (how much light is received by each surface point). The shaded color is equal to the original color with its intensity modified by this illumination. The UgRainbow program uses the same basic method, except that all three components of the original color are interpolated as well. With the chosen non-linear mapping between hue and wavelength, the interpolation of wavelengths will yield different results from the interpolation of hues. The wavelength interpolation method was implemented as an option to UgDisp. After the color for each pixel is determined, its wavelength is used to find the attenuation factor (in case of the hue interpolation, the mapping is needed to yield the wavelength for the same purpose). The attenuated color in which the wavelength is transformed to the hue is passed to the device driver for display.

5. Relativistic Ray-Tracing

Ray-tracing is a simple but slow algorithm that can display realistic computer-generated images [Whi80]. Many optimization techniques have been developed to speed up ray-tracing, but in this project we are only concerned with the question how ray-tracing is applied to relativistic scenes.

The idea of ray-tracing is to determine the intensity for each pixel, one at a time, by following a ray from the eyepoint backwards through the pixel into the scene space. The intersection of the ray with every object in the scene is then calculated. If none is encountered, then the pixel is assigned a background color. If the ray intersects more than one object, then the intersection closest to the eye will determine the color of the pixel.

Once the first intersection points have been identified for each pixel, ray-tracing allows us to calculate realistic shading effects using the same mechanisms. As shown in figure 5.1, the light intensity I, passed to the eyepoint from a point on the surface, consists of three basic components : the specular reflection S, the transmission T, and a diffused component D (not shown in the figure). The diffuse reflection can be computed using a conventional shading model [Gou71, Pho75, Bli77]. Shadows can be calculated by generating new rays $\vec{L}_i$ between the visible surface point and each light source. If these rays intersect any opaque object before they reach the light, then the surface point is in shadow with respect to the corresponding light source, and that light source will contribute no intensity to the surface point.

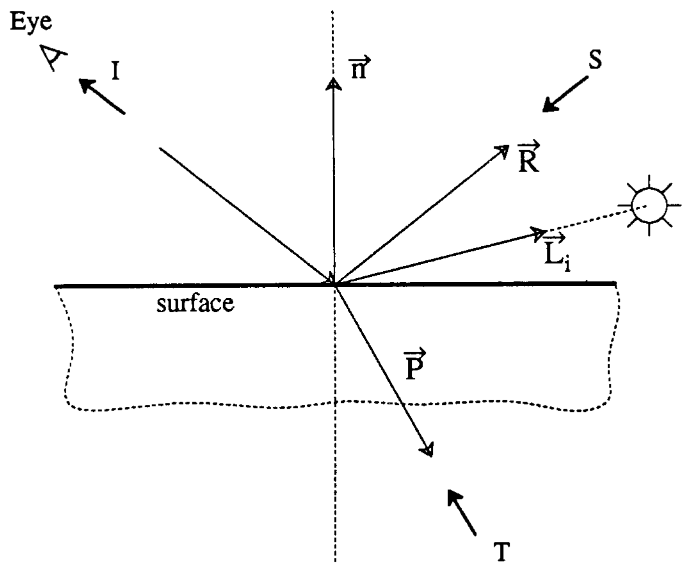


Figure 5.1

If the surface is reflective, the reflection S can be calculated by creating a new ray in the reflection direction $\vec{R}$. This direction can be determined by the simple rule that the angle of reflection must equal the angle of incidence, and that it must lie in the same plane with the observer ray and the surface normal. Similarly, if the surface is transparent, the transmission T can be computed by casting a new ray in the refraction direction $\vec{P}$. The direction of this ray must obey Snell's law.

Note that the reflection and refraction rays can be viewed as new observer rays equivalent to the the original ray, so the process of ray-tracing can be continued recursively until none of the new rays intersects any transparent or reflective objects, or the new ray is expected to contribute an insignificant amount to the surface point. The latter condition arises due to the fact that the light intensity is attenuated by reflection, refraction, and also by traveling through an absorbing medium. In a practical program, one often simply limits the depth of recursion.

To consider relativistic effects, conventional ray-tracing must be modified somewhat. One approach is to generate an event scene in a 4D space where the 4th dimension is event time. The rays which are cast backwards in time dimension can be traced through this 4D space just as the rays in the ordinary ray-tracing in 3D space. This method was attempted at first. However, it is rather inefficient to search a 4D space for intersection points, when all objects travel at more or less constant speed. Moreover, it requires complicated data structures to represent arbitrary event scenes.

A more efficient approach is to trace the rays through the 3D space scene at the proper speed corresponding to the light velocity in the particular medium and to analyze the intersections with various moving objects. The rays can only intersect the objects at the points where the ray time and the object time match exactly. In ordinary ray-tracing, it is usually easier to compute the intersections by using some parametric value t. This value is also used to solve the hidden surface problem, as the closest intersection point must have the smallest value of t. In our relativistic approach, the parameter t is replaced by a variable that records the time of the intersection (the two parameters are not quite the same as will be seen in section 5.1) and it is used to solve the ray intersection problem in a way similar to the normal ray-tracing approach.

According to the special relativity theory, an observer at a stationary eyepoint does not see equal incident and reflected angles when light is reflected from a moving object. However, for a person on the moving object the reflection law are still obeyed, and this fact is exploited to compute the reflection direction. Similar statement can be made about refracted light.

The shading computation must also take into account relativistic effects. When a vision ray interacts with a moving surface, the relative velocities are calculated and used to compute the Doppler shift and the changes in photon flux.

The motions of objects in the scene are expressed with respect to the coordinate frame of the camera (eyepoint). Thus the proper Lorentz's transformation (section 2.1), resulting in object distortions and time dilation effects, can be carried out easily in this reference frame; this is done right after the 3D space scene is read in.

5.1. Ray/Polygon Intersection

In the scope of this project, different polygons may travel at different velocities, but all vertices of the same polygon are restricted to have the same constant velocity (i.e., polygons cannot rotate). It is conceptually not too hard to extend this approach to the general case, and possible extensions will be discussed later. The face statement of UniGrafix language is extended to take an optional velocity vector whose default value is $< 0,0,0 >$ (Appendix B).

Another assumption should be addressed here. The coordinates of objects in the scene read in from the input are assumed to be those at time $t = 0$ when the shutter of the camera briefly opens.

The ray/polygon intersection calculation consists of two steps. First, the intersection event (x,y,z,t) of the ray with the moving plane containing the polygon is computed. Second, the point of intersection is determined to be inside or outside of the polygon located at its position in the past that corresponds to the intersection time.

5.1.1. Ray/Plane Intersection

For simplicity, let us first consider the primary rays which originate from the eyepoint, then extend the discussion to the general rays.

Let $\vec{R}_0$ be the position of the eyepoint (it is usually convenient to make $\vec{R}_0 = <0,0,0>$, i.e., perform a world-to-eye-transformation). And let $\vec{R}_d$ be the velocity vector of the photons that eventually reach the eyepoint at time $t = 0$. The direction of $\vec{R}_d$ is from a given pixel to the eyepoint and its magnitude is c, the speed of light in the medium considered (see figure 5.1.1.1).

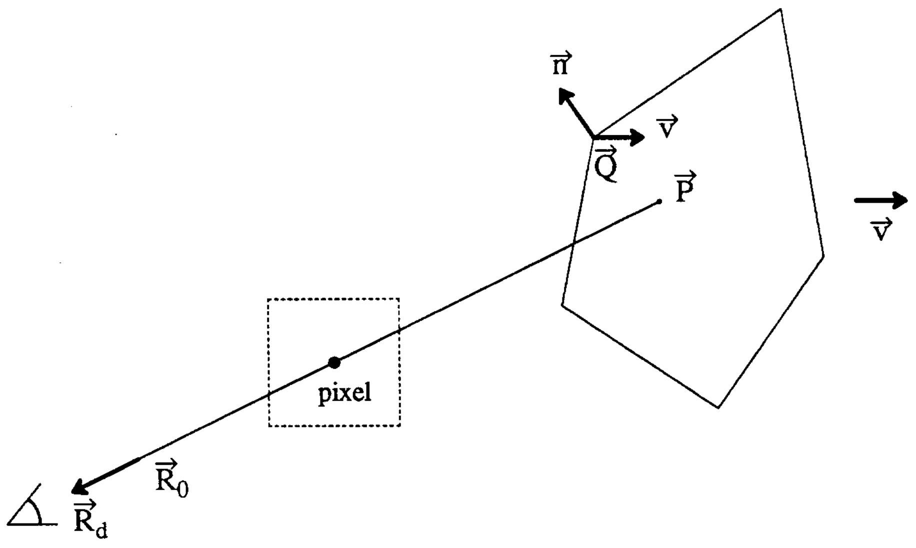


Figure 5.1.1.1

If $\vec{P}$ is the intersection point of the ray with the moving plane, and t is the intersection time, or the time in the past when the photon, which will eventually make to the shutter, is emitted from the polygon, then

$$\vec{P} = \vec{R}_0 + t\vec{R}_d, \quad t \leq 0 \qquad \{5.1.1.1\}$$

Let $\vec{Q}$ be a point on the plane at the time the photon is emitted, then

$$(\vec{P} - \vec{Q}) * \vec{n} = 0 \qquad \{5.1.1.2\}$$

where $\vec{n}$ is the normal vector of the plane which is constant in time in our model. It should be noted that since $\vec{n}$ is the normal vector of the polygon when it is at rest, $\vec{n}$ must be calculated before the Lorentz transformation is applied to the vertices of the polygon.

Arbitrarily, we choose $\vec{Q}$ to be one of the polygon's vertices:

$$\vec{Q} = \vec{Q}_0 + t\vec{v} \qquad \{5.1.1.3\}$$

Here, $\vec{Q}_0$ is some vertex which is read in directly from the scene description.

Substituting {5.1.1.1} and {5.1.1.3} into {5.1.1.2}, we obtain

$$(\vec{R}_0 + t\vec{R}_d) * \vec{n} = (\vec{Q}_0 + t\vec{v}) * \vec{n}$$

$$=> \quad t = \frac{(\vec{Q}_0 - \vec{R}_0) * \vec{n}}{(\vec{R}_d - \vec{v}) * \vec{n}}$$

If the denominator of this equation is zero, then no intersection is possible. The intersection test can also be aborted if $t \geq 0$ (the intersection is behind the eye), or if t is smaller than the largest value of t computed for previously tested polygons that intersect the ray, since the intersection point is hidden by some other polygon. Otherwise, t can be substituted into equation {5.1.1.2} to yield the coordinates of the intersection. The value of t is then saved, because it will be needed later when the secondary rays are generated (see below).

Now, let us consider a general ray. $\vec{R}_0$, $\vec{R}_d$, $\vec{P}$, $\vec{Q}$, $\vec{Q}_0$, $\vec{n}$, $\vec{v}$ are defined as before. Furthermore, let t_0 be the time associated with $\vec{R}_0$. It is the time when the photons reach the point $\vec{R}_0$ (it is 0 in the case of a primary ray). For the general ray, only $\vec{P}$ is different from its previous equation. It is now

$$\vec{P} = \vec{R}_0 + (t - t_0)\vec{R}_d\,, \quad t \leq 0 \qquad \{5.1.1.4\}$$

Solving for t by substituting {5.1.1.3} and {5.1.1.4} into {5.1.1.2}, we obtain

$$t = \frac{(\vec{Q}_0 - \vec{R}_0 + t_0\vec{R}_d) * \vec{n}}{(\vec{R}_d - \vec{v}) * \vec{n}} \qquad \{5.1.1.5\}$$

5.1.2. The Inside/Outside Test

Once we find that a ray intersects the plane of a moving polygon, its intersection time is used to calculate the position of edges of the polygon at that particular time. Then we determine whether the intersection point is inside or outside these edges of the polygon.

The testing method employed in this project requires the projection of the polygon and the intersection point onto some plane where the inside/outside test can be performed conveniently. The projected plane is one of xy, xz, and yz planes. To reduce numerical error, the plane in which the projected polygon covers the most area is chosen. This can be done by examining the three components of the polygon's normal vector. Once the polygon and the intersection point have been projected onto one of the three major coordinate planes, a half line is generated from the projected intersection point (the direction is arbitrarily chosen to be +x, +y, or +z if the projection plane is xy, yz, or xz, respectively). The inside/outside test can then be performed simply by counting the number of times the projected edges of the polygon cross the half line. If the number of crossings is odd, the point is inside the polygon; otherwise, it is outside [Har87].

The inside/outside test can be speeded up [Mar87], but as mentioned earlier, efficiency is of little concern in this project.

5.2. Shading

The shading method plays a major role in determining the quality and realism of the final image. Once an intersection has been identified to be visible, its intensity and color have to be calculated. There exists a wide variety of shading models and they usually exhibit a trade-off between realism and efficiency. In this project, a basic shading method often used in ray-tracers is employed: a Phong shading model enhanced with the necessary rays to model shadows, reflections, and refractions. In this section, certain techniques which are quite different from normal ray-tracing will be emphasized, e.g., how to determine the direction of reflected or refracted rays.

5.2.1. Smooth Shading

There are two basic strategies for obtaining smooth shading: Gouraud shading [Gou71] and Phong shading [Pho75]. They are briefly outlined as follows:

In the Gouraud shading method, a vertex normal is calculated at each vertex as a function of the surface normal vectors of all polygons that contain the vertex. Then for all polygons in the scene, the vertex normal at each vertex is used to calculate the vertex shade. The shade of a given point within a polygon is then determined by double interpolation of the vertex shades: once along the edges, and once between two points on two edges (e.g., on the same scan-line).

In the Phong shading method, vertex normals are computed as above, but the surface normal vector itself is interpolated instead of the shading value. Thus, in order to calculate the shade of a given point, its surface normal must be computed before the shade can be determined.

The Phong shading model was chosen in this project for a number of reasons. First, care must be taken in the use of the Gouraud shading when one vertex is in shadow and another is not. The linear interpolation is also incorrect if a surface reflects the light source. Moreover, if reflections and refractions are modeled, the surface normals at intersection points have to be calculated anyway. Since the Gouraud shading method is simpler and faster than Phong's, it is most appropriate when the surfaces are not reflective, and when shadows are ignored. This can be easily added to the existing program.

Because the objects in the scene are restricted to have constant velocity, all surface normals are also constant with time. The vertex normals, therefore, can be preprocessed before the intersection times are calculated. After the scene description is read in, all vertex normals are computed by averaging the surface normals of all the faces that contain the vertex. The weighting factor for each face is the angle of the face corner at that vertex. When an intersection point is found, it is always mapped to its original position in the scene description (i.e., its position at t = 0) where the linear interpolation of normal components can be conveniently performed.

5.2.2 Direct Illumination by Directional Light

When the diffusely and specularly reflected components are computed, each light source in the scene must be considered and their results summed up [Pho75]. In this project, directional light sources are modeled as infinitely distant point light sources, and the implementation of the local light sources is left for future work.

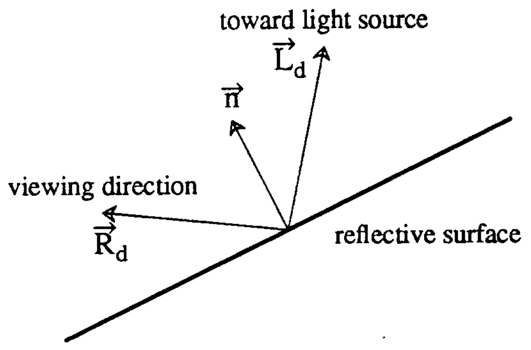


Figure 5.2.2.1

A light source contributes to the light intensity reflected from a surface in the viewing direction if it illuminates the same side of the surface that the viewing ray hits (figure 5.2.2.1). In ordinary ray-tracing, the sign of the dot product between the ray vector $\vec{R}_d$ and the surface normal $\vec{n}$ (i.e., $\vec{R}_d * \vec{n}$) is computed. If the dot product between $\vec{n}$ and a vector toward the light source $\vec{L}_d$ (i.e., $\vec{L}_d * \vec{n}$) generates the same sign, the light source illuminates the surface. If the opposite sign is generated, then the light source can only illuminate the opposite side of the surface, and the light source contributes no intensity to the surface.

In relativistic ray-tracing, some modifications to the above technique are needed. A photon which is moving in the direction of a viewing ray $\vec{R}_d$ is observed to be moving in some other direction $\vec{R}$ with respect to a person on the surface that travels with velocity $\vec{v}$ (figure 5.2.2.2). In other words, the photon must be ejected in the $\vec{R}$ direction with respect to the moving surface in order to travel in the direction of the viewing ray $\vec{R}_d$. Thus, the dot product $\vec{R} * \vec{n}$ is used in place of $(\vec{R}_d * \vec{n})$. Similarly, the vector toward the light source $\vec{L}_d$ is observed by a person on the moving surface as some vector $\vec{L}$. In summary, if the sign of $\vec{R} * \vec{n}$ is the same as $\vec{L} * \vec{n}$, then the light source illuminates the surface; otherwise, the light source is ignored in the direct illumination calculation.

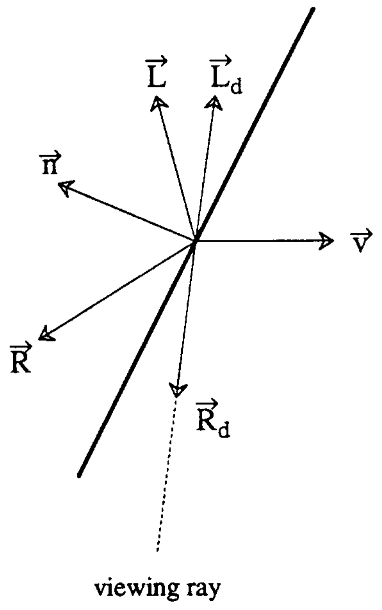


Figure 5.2.2.2

The vectors $\vec{R}$ and $\vec{L}$ can be obtained by using the Lorentz velocity addition [Eis81] which will be explained in detail in Section 5.2.4. Here is an overview.

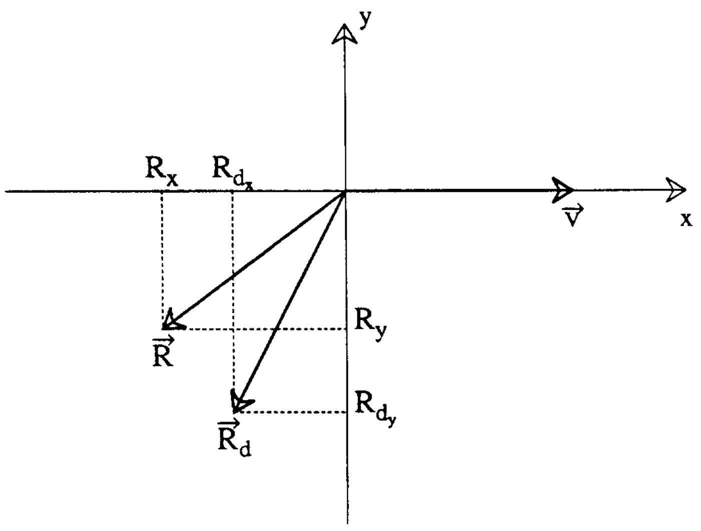


Figure 5.2.2.3

Let's define a 2D Cartesian coordinate system xy in the plane containing vectors $\vec{R}_d$ and $\vec{v}$ in such a way that the vector $\vec{v}$ is in +x direction (figure 5.2.2.3). The vector $\vec{R}$ in this coordinate system has the components [Eis81]

$$R_x = \frac{R_{d_x} - v}{1 - vR_{d_x}/c^2} \qquad \{5.2.2.1\}$$

$$R_y = \frac{\sqrt{1 - v^2/c^2}}{1 - vR_{d_x}/c^2} R_{d_y} \qquad \{5.2.2.2\}$$

This calculation is extremely time consuming. Since $\vec{R}_d$ and $\vec{v}$ are expressed in eye coordinates, it requires some transformation into the 2D coordinates just mentioned. Then, after the vector $\vec{R}$ in this coordinate system is computed, a projection back into the eye coordinate system is required. Note that if $v << c$ then the square root term in formula {5.2.2.2} can be omitted and vector $\vec{R}$ will become

$$R_x = \frac{1}{1 - vR_{d_x}/c^2}(R_{d_x} - v)$$

$$R_y = \frac{1}{1 - vR_{d_x}/c^2} R_{d_y}$$

Vector $\vec{R}$ now has the same direction as $(\vec{R}_d - \vec{v})$ as expected from Galilean relativity theory. Thus if $v << c$, the classical vector $(\vec{R}_d - \vec{v})$ is used instead in order to reduce the running time.

5.2.3. Shadows

As mentioned earlier, when the diffuse and specular reflections are computed, each light source in the scene must be considered. If, however, a surface area is in shadow with respect to some light source, then that light source can be eliminated from the lighting calculation of this area. That is because that light source contributes no direct intensity to the area.

A surface point is illuminated by some light source if and only if the photon that is ejected from the light source towards the surface point and that is expected to eventually make through the camera shutter is not blocked in the past by some object (possibly moving).

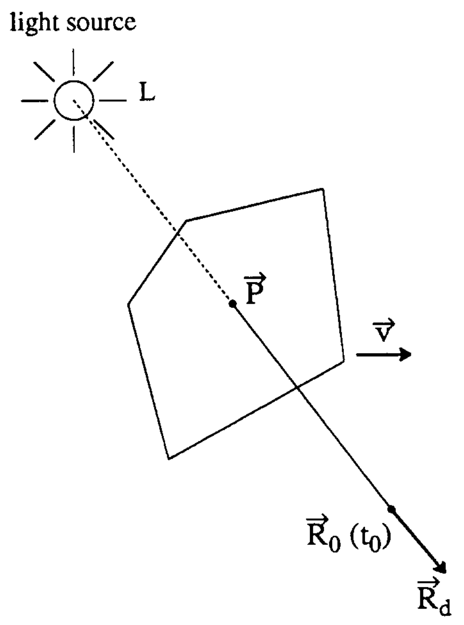


Figure 5.2.3.1

Figure 5.2.3.1 shows point $\vec{R}_0$ in shadow with respect to the light source L. During the ray-tracing process, the point $\vec{R}_0$ is somehow encountered by an observer ray and its illumination needs to be determined. Let t_0 be the intersection time corresponding to the intersection point $\vec{R}_0$. (Note that the photon that reaches $\vec{R}_0$ at time t_0 is expected to make it through the shutter). To determine if $\vec{R}_0$ is illuminated by L, the shadow ray from $\vec{R}_0$ towards the light source L is cast backwards in time starting from t_0. Equation {5.1.1.5} and the inside/outside test are used to decide if it intersects any polygon in the scene. $\vec{R}_0$ is in shadow if such an intersection is found and if the intersected polygon represents an opaque surface.

If the scene contains transparent objects, there exists a problem that is also encountered in ordinary ray-tracing. If the shadow ray intersects a transparent object then the ray is bent by the object's refractive material. The bending pulls the shadow ray away from the direction of the light source; therefore, it is no longer the actual "shadow" ray (figure 5.2.3.2). The correction of this problem represents an almost impossible computational expense. The program written for this project simply ignores this problem and propagates the shadow ray without bending.

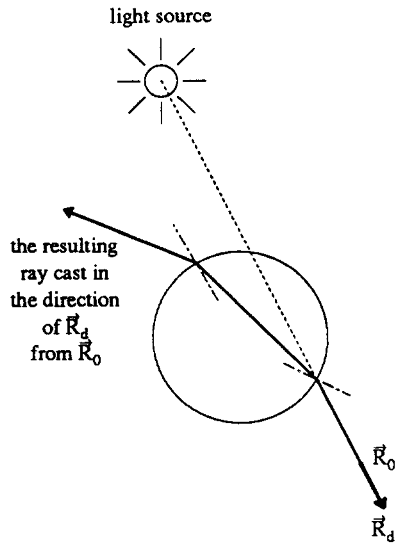

light source
the resulting ray cast in the direction of $\vec{R}_d$ from $\vec{R}_0$
$\vec{R}_0$
$\vec{R}_d$

Figure 5.2.3.2

Yet another simplification concerns the propagation of light rays through absorbing media. Theoretically, the light intensity must by attenuated according to its traveling distance within the transparent object. For this project this volume absorption has been ignored, and the light intensity is attenuated only at the surface of a transparent object by the coefficient of transmission k_t.

5.2.4. Specular Reflection From Other Surfaces

This section does not discuss how to calculate specular reflection from a light source. Such discussion can be found in [Pho75]. This section however focuses on the specular reflection of other objects.

During the ray-tracing process, if a viewing ray intersects a reflective surface then the reflection in this surface can be modeled by tracing a new ray in the direction of reflection. If the new ray intersects a second surface, this surface will be seen reflected in the first surface. Multiple reflections can be modeled by repeating this process recursively.

If the color returned by the reflected ray is C, then the reflection contributes to the reflective surface an amount of Ck_s, where k_s is the reflection coefficient that specifies what fraction of incident light is reflected. This is the same as the specular reflection coefficient in the Phong shading model. The value of k_s depends on the angle of incidence [Pho75], but for simplicity, we use a fixed value.

The major difference from normal ray-tracing in modeling the specular reflection of an object is how the reflection direction is calculated. In non-relativistic ray-tracing, the reflective surface has the property that the angle of incidence is equal to the angle of reflection. This is not true for an outside observer watching a surface moving at relativistic speed.

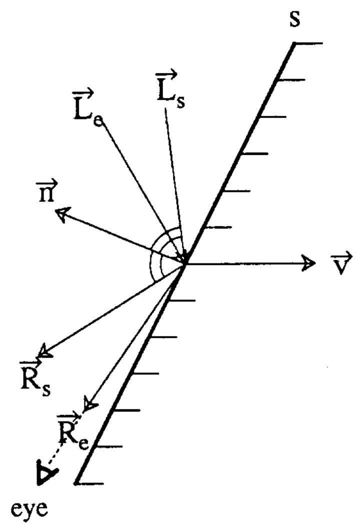


Figure 5.2.4.1

Figure 5.2.4.1 shows a reflective surface s moving with velocity $\vec{v}$. The light vector $\vec{L}_e$ incident to s will be seen as vector $\vec{L}_s$ to an observer on the moving surface. According to Einstein's first postulate, the reflection law is still observed in the inertial frame of reference associated with the surface itself [For74]. In other words, the reflected ray $\vec{R}_s$ seen by the observer on surface s has the angle of reflection the same as the angle of incidence from vector $\vec{L}_s$. Vector $\vec{R}_e$ which is the reflected ray in the eye coordinates is, in fact, the vector $\vec{R}_s$ but in a different frame of reference. The mapping between $\vec{L}_s$ vs. $\vec{L}_e$ and $\vec{R}_s$ vs. $\vec{R}_e$ is called Lorentz velocity transformation [Eis81] and is described below.

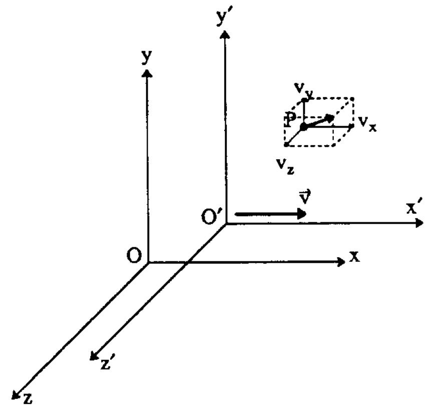


Figure 5.2.4.2

Figure 5.2.4.2 shows two observers O and O′ with the primed observer moving relative to the unprimed observer at a constant velocity v. For simplicity, we choose the two frames of reference in such a way that the axes x, y, z and x′, y′, z′ are mutually parallel, and the relative velocity $\vec{v}$ is in the +x direction.

If $<v_x, v_y, v_z>$ and $<v'_x, v'_y, v'_z>$ are the velocities of some object P measured in the unprimed frame and the primed frame, respectively, then they are related according to the following formulas:

$$v'_x = \frac{v_x - v}{1 - vv_x/c^2} \qquad \{5.2.4.1\}$$

$$v'_y = \frac{\sqrt{1 - v^2/c^2}}{1 - vv_x/c^2} v_y \qquad \{5.2.4.2\}$$

$$v'_z = \frac{\sqrt{1 - v^2/c^2}}{1 - vv_x/c^2} v_z \qquad \{5.2.4.3\}$$

Equivalently,

$$v_x = \frac{v'_x + v}{1 + vv'_x/c^2} \qquad \{5.2.4.4\}$$

$$v_y = \frac{\sqrt{1 - v^2/c^2}}{1 + vv'_x/c^2} v'_y \qquad \{5.2.4.5\}$$

$$v_z = \frac{\sqrt{1 - v^2/c^2}}{1 + vv'_x/c^2} v'_z \qquad \{5.2.4.6\}$$

Note that the above formulas require a special arrangement of the two frames of reference. In particular, their +x and +x′ direction must coincide with the direction of velocity $\vec{v}$. Let us call the transformation performing this task T_v. The calculation of the reflection direction then consists of these steps:

1) Apply transformation T_v on vector $\vec{R}_e$

2) Use formulas {5.2.4.1} through {5.2.4.3} to compute $\vec{R}_s$.

3) Apply reflection law to compute $\vec{L}_s$ from $\vec{R}_s$

4) Use formulas {5.2.4.4} through {5.2.4.6} to compute $\vec{L}_e$

5) Apply inverse transformation T_v^{-1} on vector $\vec{L}_e$

In order to speed up this calculation, these five steps are expressed in terms of matrix multiplications, then combined into a single transformation matrix.

Formulas {5.2.4.1} through {5.2.4.3} can be expressed in homogeneous coordinates as

$$\begin{bmatrix} v'_x & v'_y & v'_z & v'_t \end{bmatrix} = \begin{bmatrix} v_x & v_y & v_z & v_t \end{bmatrix} M_1$$

where

$$M_1 = \begin{bmatrix} 1 & 0 & 0 & -v/c^2 \\ 0 & \sqrt{1-v^2/c^2} & 0 & 0 \\ 0 & 0 & \sqrt{1-v^2/c^2} & 0 \\ -v & 0 & 0 & 1 \end{bmatrix} \qquad \{5.2.4.7\}$$

Similarly, formulas {5.2.4.4} through {5.2.4.6} can be expressed as

$$\begin{bmatrix} v_x & v_y & v_z & v_t \end{bmatrix} = \begin{bmatrix} v'_x & v'_y & v'_z & v'_t \end{bmatrix} M_2$$

where

$$M_2 = \begin{bmatrix} 1 & 0 & 0 & v/c^2 \\ 0 & \sqrt{1-v^2/c^2} & 0 & 0 \\ 0 & 0 & \sqrt{1-v^2/c^2} & 0 \\ v & 0 & 0 & 1 \end{bmatrix} \qquad \{5.2.4.8\}$$

Note that $M_1M_2 = (1 - v^2/c^2)I$ (identity matrix). This is no surprise, because the same vector is obtained if all its components in homogeneous coordinates are multiplied by the same factor (in this case $(1 - v^2/c^2)$), and because M_1 and M_2 are inverses of each other.

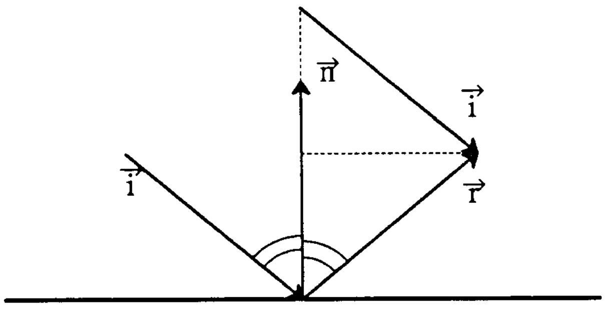


Figure 5.2.4.3

Now, let's consider the transformation for step 3. Let $\vec{n}$ be the **unit** normal vector of the reflective surface s. Define $\vec{i}$ and $\vec{r}$ to be the incidence and reflection rays, respectively (figure 5.2.4.3), then

$$\vec{i} = \vec{r} - 2(\vec{r} * \vec{n})\vec{n} \qquad \{5.2.4.9\}$$

Equation {5.2.4.9} **can be** rewritten as

$$\begin{cases} i_x = r_x - 2(n_x r_x + n_y r_y + n_z r_z)n_x \\ i_y = r_y - 2(n_x r_x + n_y r_y + n_z r_z)n_y \\ i_z = r_z - 2(n_x r_x + n_y r_y + n_z r_z)n_z \end{cases}$$

In matrix form, it is

$$\vec{i} = \vec{r}R$$

where

$$R = \begin{bmatrix} 1-2n_x^2 & -2n_x n_y & -2n_x n_z & 0 \\ -2n_x n_y & 1-2n_y^2 & -2n_y n_z & 0 \\ -2n_x n_z & -2n_y n_z & 1-2n_z^2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

It is important to note that the vector $\vec{n}$ must be measured in the uwt coordinates of the moving surface. If $\vec{n}_e$ is the surface normal in eye coordinates, then its vector in uwt coordinates is $\vec{n} = \vec{n}_e T_v$. Moreover, since the vector $\vec{n}_e$ is the normal vector of the polygon when it is at rest, $\vec{n}_e$ should be calculated before the Lorentz transformation is applied to the vertices of the polygon.

Finally, let's compute the transformation matrix T_v which brings the velocity vector $\vec{v}$ into the +x direction. This requires two rotations; however, there is a better way to compute T_v. The computation is exactly as that shown in section 4.1.1.1. Matrix T_v is the same as matrix R in {4.1.1.1.1} and is recalled here :

$$T_v = \begin{bmatrix} u_x & w_x & t_x & 0 \\ u_y & w_y & t_y & 0 \\ u_z & w_z & t_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \{5.2.4.10\}$$

where vectors $\vec{u}$, $\vec{w}$, $\vec{t}$ are given in equations {4.1.1.1.2},{4.1.1.1.3}, and {4.1.1.1.4} (or in case $\vec{v}$ and $\vec{z}$ are parallel, equations {4.1.1.1.5}, {4.1.1.1.6}, and {4.1.1.1.7})

Because T_v is an orthogonal matrix, its inverse is simply its transpose.

$$T_v^{-1} = T_v^T \qquad \{5.2.4.11\}$$

The direction $\vec{L}_e$ can now be computed from vector $\vec{R}_e$:

$$\vec{L}_e = \vec{R}_e \,.\, T_v \,.\, M_1 \,.\, R \,.\, M_2 \,.\, T_v^T$$

While matrices T_v, M_1, and M_2 depend only on $\vec{v}$, matrix R depends on both $\vec{v}$ and $\vec{n}$ - the surface normal vector in eye coordinates. For each surface in the scene, these matrices are clearly defined; therefore, they can be computed and then multiplied into a single matrix, $M = T_v \,.\, M_1 \,.\, R \,.\, M_2 \,.\, T_v^T$, during polygon preprocessing. M is then available for fast calculation of the reflection direction.

5.2.5. Refraction

If a viewing ray intersects a transparent object then the refraction through this object can be modeled by casting a new ray in the direction of refraction. If the color returned by this ray is C, then the transmitted light contributes an amount of intensity Ck_tc_s, where k_t is the transmission coefficient and c_s is the color of the intersected object. The value of k_t depends on the incident angle, and in some cases, the index of refraction of the transparent material. However, for simplicity, we use a fixed value. Also note that no volume attenuation is performed in this project.

The direction of the transmitted ray is computed in a manner similar to that of the reflected ray. If $\vec{T}_e$ is the vector of the viewing ray then the vector $\vec{L}_e$ of the transmitted ray is

$$\vec{L}_e = \vec{T}_e \,.\, T_v \,.\, M_1 \,.\, R \,.\, M_2 \,.\, T_v^T$$

where T_v, M_1, M_2 are the matrices defined in the last section. The matrix R is discussed here.

The matrix R depends on the normal vector $\vec{n}$ of the surface and the indices of refraction n_T and n_L of the mediums of the viewing ray and the transmitted ray, respectively.

If $n_T = n_L$ then the transmitted ray vector is the same as the incident viewing ray vector. Thus $R = I$ in this case.

If $n_T \neq n_L$ then we observe Snell's law (figure 5.2.5.1).

$$\frac{\sin\alpha_T}{\sin\alpha_L} = \frac{n_L}{n_T} \qquad \{5.2.5.1\}$$

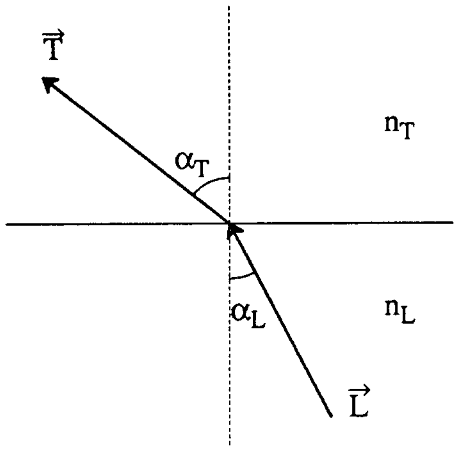


Figure 5.2.5.1

Since the sin terms in the equation {5.2.5.1} are nonlinear, there is no single matrix R that can perform the required transformation. The formula {5.2.5.1} must be used to compute vector $\vec{L}$ from vector $\vec{T}$. Because light travels at different speeds in mediums with different indices of refraction, the factor $(\frac{n_L}{n_T})$ is used to adjust its magnitude (i.e., to adjust vector $\vec{L}$).

5.2.6. Doppler Effect and Photon Flux

When photons reflect (or refract) between moving surfaces, Doppler effect and photon flux changes must be calculated for each interaction of all rays that we trace. Suppose that an observer ray passes through some point $\vec{R}_0$ and points into direction $\vec{R}_d$. Suppose further that this ray intersects some surface at point $\vec{P}$ (if no intersection is found, the ray can be discarded), then the color returned by the ray must be modified to take into account the Doppler effect and the photon flux change. In detail, the relative velocity of $\vec{P}$ and $\vec{R}_0$ is calculated, then used to adjust the hue and brightness of the returned color. The adjustment computation can be found in chapter 3. The calculation of the relative velocity is very similar to what described in section 5.2.4. Let $\vec{v}_e$ be the velocity of point $\vec{P}$ and $\vec{v}$ be the velocity of point $\vec{R}_0$ measured in the eye coordinates (figure 5.2.6.1). Then the velocity $\vec{v}_r$ of point $\vec{P}$ seen by an observer at $\vec{R}_0$ is

$$\vec{v}_r = \vec{v}_e \cdot T_v \cdot M_1 \cdot T_v^T$$

where $\vec{T}_v$ and M_1 are matrices {5.2.4.10} and {5.2.4.7} of section 5.2.4.

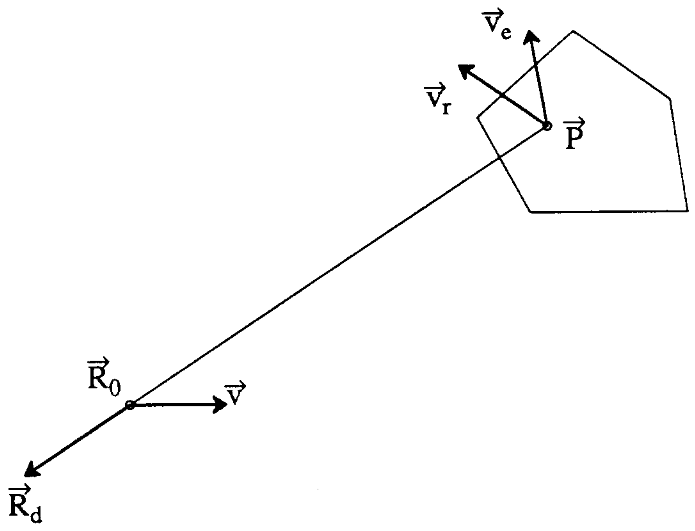


Figure 5.2.6.1

6. Conclusions and Future Improvements

Some relativistic effects experienced by objects moving at high speed were described and discussed. Rendering techniques that implement these effects were also presented. In particular, two relativistic renderers were developed to allow the user to "see" what objects passing at high velocity look like.

Possible future improvements and extensions of these renderers are briefly discussed below. We only present those techniques unique to the relativistic rendering process and ignore enhancements such as anti-aliasing, good shading model, etc,... found in standard rendering processes.

If objects are allowed to move in arbitrary ways, including rotations and even accelerations, the simple formulas derived in chapter 2 and chapter 5 are no longer valid. The Lorentz contraction turns out to be complicated because no fixed frame of reference can be assigned to a particular object. The amount of contraction is a function of time just as the velocity is. The distortion derived in section 2.2 and 5.1.1, however, needs just a little modification. The moving path of a point on the object is no longer a line (equation {2.2.4} and {5.1.1.3}) but some other time dependent path. The intersection point now lies on a possibly curved surface, and the equations to be solved are more complicated requiring iterative numerical methods. Doppler effects and intensity changes are easily calculated once the instantaneous velocity at the intersection point is deduced from the intersection time.

The introduction to chapter 5 mentioned another possible approach to relativistic ray-tracing. It suggests that a 4D space scene is created, and the rays are traced in this scene just as rays in the ordinary 3D ray-tracer. This idea was rejected mainly because our data structures were too complicated for the simple case of linear motion. For the general case of arbitrary motions, this 4D ray-tracing approach looks more attractive and should therefore be investigated further.

As mentioned from time to time in this report, the running times of these renderers were of little concern. The renderer implemented as a preprocessor runs sufficiently fast, and there seems to be no opportunity to greatly improve it. The relativistic ray-tracer, on the other hand, can be speeded up significantly. Techniques routinely used to improve the running time of ordinary ray-tracing can also be applied here with some precautions. One of the Berkeley UniGrafix renderers, UgRay employs a uniform spatial subdivision, which cuts the computation cost of ray-tracing dramatically [Mar87]. The same basic approach can be applied here.

In brief, the uniform spatial subdivision algorithm subdivides the space containing a scene into small regions. When a ray enters a region, intersection tests are performed against all primitives (e.g., polygons) that are intersecting that region. If no intersections are found, the ray is propagated into the next region intersected by the ray [Dip84]. Applying this approach to relativistic ray-tracing, a subdivision of 4D space (space and time) is an appropriate solution. These event-cells can be thought of as moving spatial regions. Thus time coordinates must be considered when the ray propagates from region to region. There is, however, a simpler technique that does not require the subdivision of 4D space. Even though it is not as fast as the above 4D method, its running time is good enough so that we have chosen to use this technique in this project; furthermore, it required very little modification to the standard ray-tracer algorithm and portions of UgRay could be used for this purpose. In this method, the subdivision is carried out in 3D as usual. The propagation of rays through the regions is also unchanged. The only change that had to be made was in the algorithm that inserts polygons into the spatial subdivision.

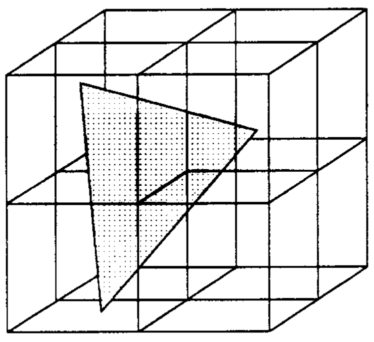


Figure 6.1

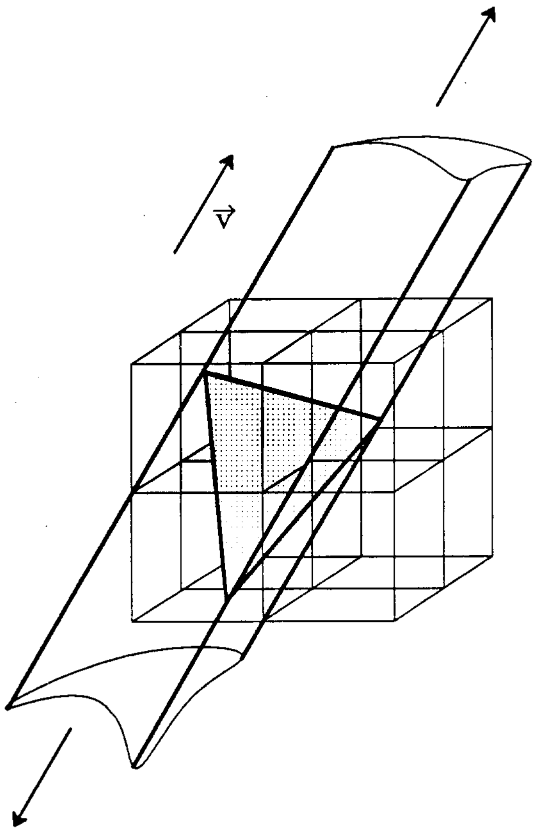


Figure 6.2

In the standard ray-tracer, the polygon P is inserted into a cell C if it touches that cell (figure 6.1). In our new method, the insertion is performed if the cell intersects with the prism swept out by the moving polygon P (figure 6.2). In order to reduce the number of ray/plane intersection tests further, for each subdivided region, we compute the minimum and maximum time coordinates where any ray originating from the eyepoint can enter and leave the region, by finding the shortest and longest distances from the eye to the sphere surrounding the cell. This time interval shortens the length of the prism that needs to be inserted into the subdivision cell structure (figure 6.3). However, if we want to consider reflected, refracted, or shadow rays, there is no bound on t_{max} since secondary ray paths can be arbitrarily complicated.

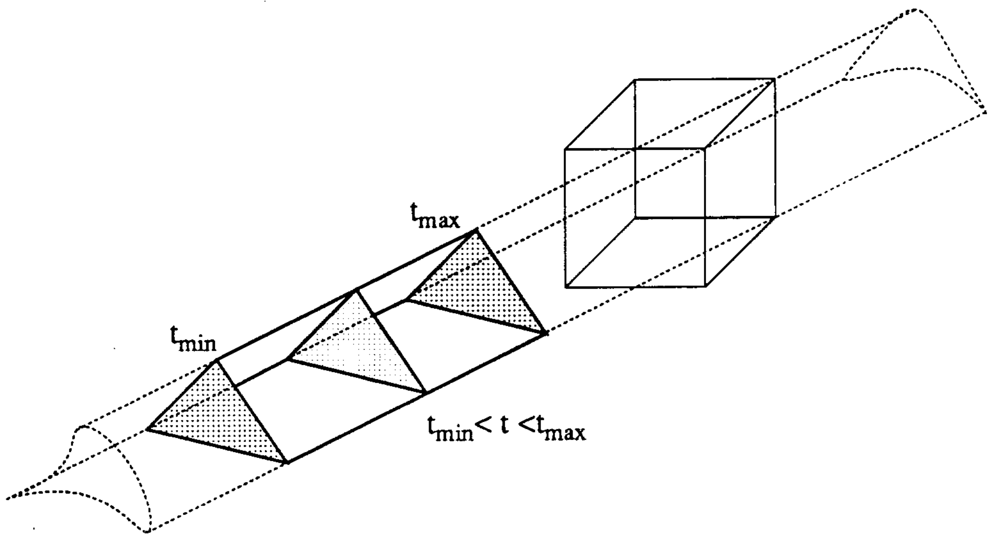


Figure 6.3

Appendix A
References

References

[Bli77] J. F. BLINN, "Models of Light Reflection for Computer Synthesized Pictures", *Computer Graphics 11*, 2 (1977), 192-198.

[Dip84] M. DIPPE and J. SWENSEN, "An Adaptive Subdivision Algorithm and Parallel Architecture for Realistic Image Synthesis", *Computer Graphics 18*, 3 (July 1984), 149-158.

[Eis81] R. M. EISBERG and L. S. LERNER, *Physics Foundations and Applications, Combined Volume*, McGraw-Hill, (1974)

[Fey63] FEYNMAN, *The Feynman Lectures on Physics, Chapter 17: Space-Time, Volume 1*, , Addison Wesley, (1963).

[For74] K. W. FORD, *Classical and Modern Physics, Volume 3*, John Wiley & Sons, (1974), 994-1029.

[Gal86] N. GAL, "Hidden Feature Removal and Display of Intersecting Objects in UNIGRAFIX", Tech. Report UCB/CSD 86/282, U.C. Berkeley, (Jan 1986).

[Gou71] H. GOURAUD, "Continuous Shading of Curved Surfaces", *IEEE Transactions, Vol. c-20, No. 6,*, (June 1971), 623-629.

[Har87] S. HARRINGTON, *Computer Graphics - A Programming Approach, Second Edition*, McGraw-Hill, (1987).

[Mar87] D. M. MARSH, "UgRay - An Efficient Ray-Tracing Renderer for UniGrafix", Tech. Report UCB/CSD 87/360, U.C. Berkeley, (May 1987).

[New79] W. M. NEWMAN and R. F. SPROUL, *Principles of Interactive Computer Graphics, Second Edition*, McGraw-Hill, (1979).

[Pho75] B. T. PHONG, "Illumination for Computer Generated Pictures", *Comm. of the ACM 18*, 6 (June 1975), 311-317.

[Seq83] C. H. SEQUIN, M. SEGAL, and P. WENSLEY, "UNIGRAFIX 2.0 User's Manual and Tutorial", Tech. Report UCB/CSD 83/161, U.C. Berkeley, (Dec 1983).

[Seq89] C. H. SEQUIN and H. P. NGUYEN, "Relativistic Rendering", Submitted to SIGGRAPH '89.

[SeS83] C. H. SEQUIN and P. S. STRAUSS, "UNIGRAFIX", *IEEE 1983 Proceedings of the 20th Design Automation Conference*, (1983), 374-381.

[Whi80] T. WHITTED, "An Improved Illumination Model for Shaded Display", *Comm. of the ACM 23*, 6 (June 1980), 343-349.

Appendix B
Manual Pages

NAME

uglorentz – compute relativistic geometry distortions of moving objects.

SYNOPSIS

uglorentz [options] [< scene]

DESCRIPTION

Uglorentz reads in a UNIGRAFIX object description and the velocity of the object (specified with option -vel), computes the Lorentz contraction and the distortion due to Time-Of-Flight Differences of the Photons, then outputs a standard UNIGRAFIX scene description of the distorted object. The output of *uglorentz* should be sent to *ugdoppler.*

INPUT

Uglorentz accepts UNIGRAFIX scene descriptions of moving objects.

OPTIONS

Options begin with a dash '–' followed by two or three lower-case letters.

-vel *x y z*
: Velocity of all objects in the scene. *x, y, z* must be expressed as the fraction of the speed of light. Default value is <0 0 0>.

-fi *input-file*
: Read input scene from file *input-file.* If this option is not specified, the standard input is read.

-fo *output-file*
: Write output to file *output-file.* If this option is not specified, the standard output is used.

SEE ALSO

ugdoppler(UG)	compute doppler effects and brightness changes of moving objects.
ugrainbow(UG)	render UNIGRAFIX scene produced by *ugdoppler.*

AUTHOR

Huu Phu Nguyen
huuphu@miro.Berkeley.EDU

NAME

ugdoppler – compute color changes of moving objects.

SYNOPSIS

ugdoppler [options] [< scene]

DESCRIPTION

Ugdoppler reads in a UNIGRAFIX object description produced by *uglorentz* and the velocity of the object (specified with option -vel), computes the Doppler effects and brightness changes, then assigns a proper color for each vertex in the scene. The output of *ugdoppler* should be sent to *ugrainbow*.

INPUT

Ugdoppler accepts UNIGRAFIX scene descriptions of moving objects.

OPTIONS

Options begin with a dash '–' followed by two or three lower-case letters.

-vel *x y z*
Velocity of all objects in the scene. *x, y, z* must be expressed as the fraction of the speed of light. Default value is <0 0 0>.

-fi *input-file*
Read input scene from file *input-file*. If this option is not specified, the standard input is read.

-fo *output-file*
Write output to file *output-file*. If this option is not specified, the standard output is used.

-fm *message-file*
Write warning messages such as 'max intensity is 1.2, clip to 1.0' to file *message-file*. If this option is not specified, messages are sent to the standard error.

-gw Write wavelength in place of hue in the output color statements. Should be used if some colors are outside of the visible range.

SEE ALSO

ugrainbow(UG)	render UNIGRAFIX scene produced by *ugdoppler*.
uglorentz(UG)	compute relativistic geometry distortions of moving objects.

AUTHOR

Huu Phu Nguyen
huuphu@miro.Berkeley.EDU

NAME

ugrainbow – a relativistic renderer for UNIGRAFIX

SYNOPSIS

ugrainbow [options] [< scene]

DESCRIPTION

Ugrainbow is a relativistic renderer for viewing UNIGRAFIX scenes consisting of objects moving at high speed. It is a derivative of *ugdisp*.

INPUT

Ugrainbow accepts UNIGRAFIX scene descriptions produced by *ugdoppler*. Scene descriptions are written in the UNIGRAFIX language with an extension to the vertex statement:

Extended Vertex Statement

The extended vertex statement accepts a color ID. It has the syntax:

v *ID x y z colorID;*

The colorID must be given or an error will occur.

OPTIONS

Ugrainbow accepts all Ugdisp's options. It understands extra options given below.

-ih Interpolate vertex colors in the hue domain, in addition to saturation, and value.

-iw Interpolate vertex colors in the wavelength domain, in addition to saturation and value. Default is -ih.

-wi Hue values in color statements read from input are interpreted as wavelengths. This option allows the user to specify invisible colors such as ultra-violet. See option **-gw** of *ugdoppler*. Option **-wi** implies option **-iw**.

-dp Write output in PostScript language format (Black and White only). Output file is POSTSCRIPTFILE. It is not sent to any PostScript printer.

BUGS

All those found in Ugdisp.

SEE ALSO

ugdisp(UG)	render a UNIGRAFIX scene on a screen or plotter.
ugdoppler(UG)	compute doppler effects and brightness changes of moving objects.
uglorentz(UG)	compute relativistic geometry distortions of moving objects.

AUTHOR

Huu Phu Nguyen
huuphu@miro.Berkeley.EDU

NAME

ugphoton – a relativistic ray-tracing renderer for UNIGRAFIX

SYNOPSIS

ugphoton [options]

DESCRIPTION

Ugphoton is a relativistic ray-tracing renderer for viewing UNIGRAFIX scenes consisting of objects moving at high speed. It is similar to *ugray*.

INPUT

Ugphoton accepts scene descriptions in *ugray* format with an extension to the face and instance statements:

Extended Face and Instance Statements

The extended face and instance statements accept an optional velocity. If the velocity is not given, it defaults to <0 0 0>. For the instance statement, the velocity is applied to all faces associated with the instance call.

f *[ID] (v1 v2 .. vn) [colorID] [illum] < x y z >;*

i *[ID] (defID [colorID] [xforms]) < x y z >;*

x, y, z must be expressed as the fraction of the speed of light.

OPTIONS

Ugphoton accepts all Ugray's options. It understands an extra option given below.

-vel *x y z*
If this option is given, all objects in the scene have velocity < x y z>. *x, y, z* must be expressed as the fraction of the speed of light.

BUGS

All those found in Ugray.

Option "**-cn** *n*" does not work (see *ugray*). *n* always takes value of -1 which means the spatial subdivision techniques can not be used.

The shading equation used is somewhat different from that of *ugray*.

SEE ALSO

ugray(UG)	a ray-tracing renderer for UNIGRAFIX.

AUTHOR

Huu Phu Nguyen
huuphu@miro.Berkeley.EDU

Appendix C
Sample Images

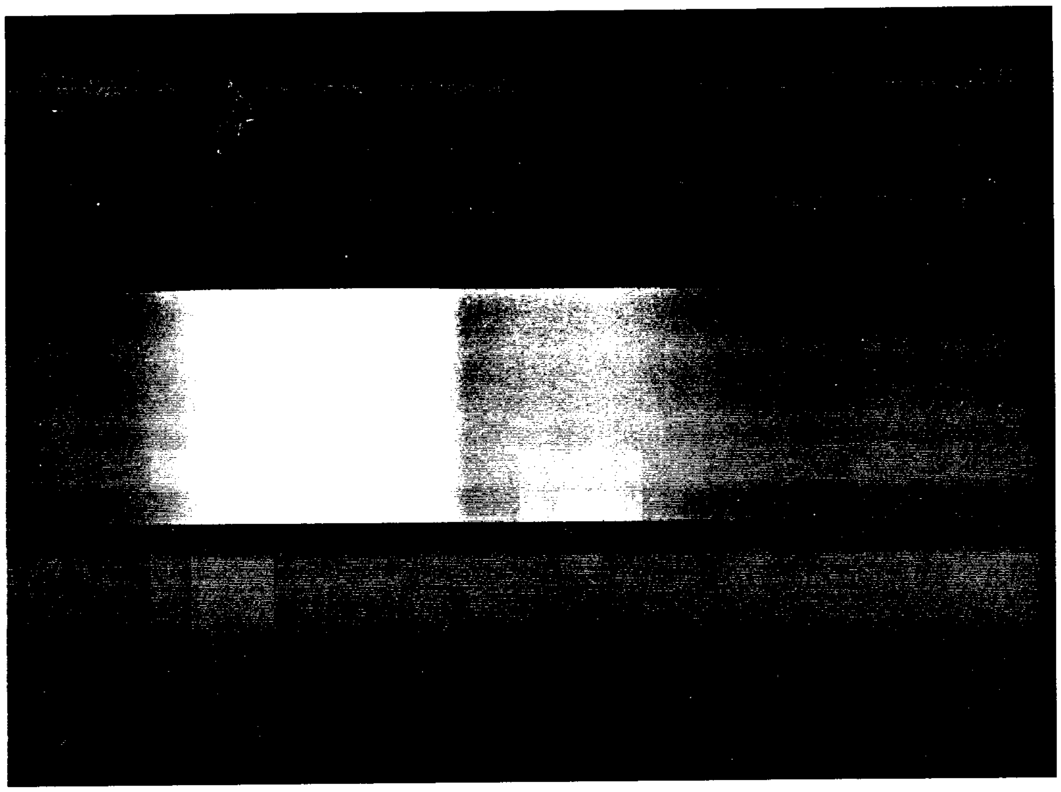
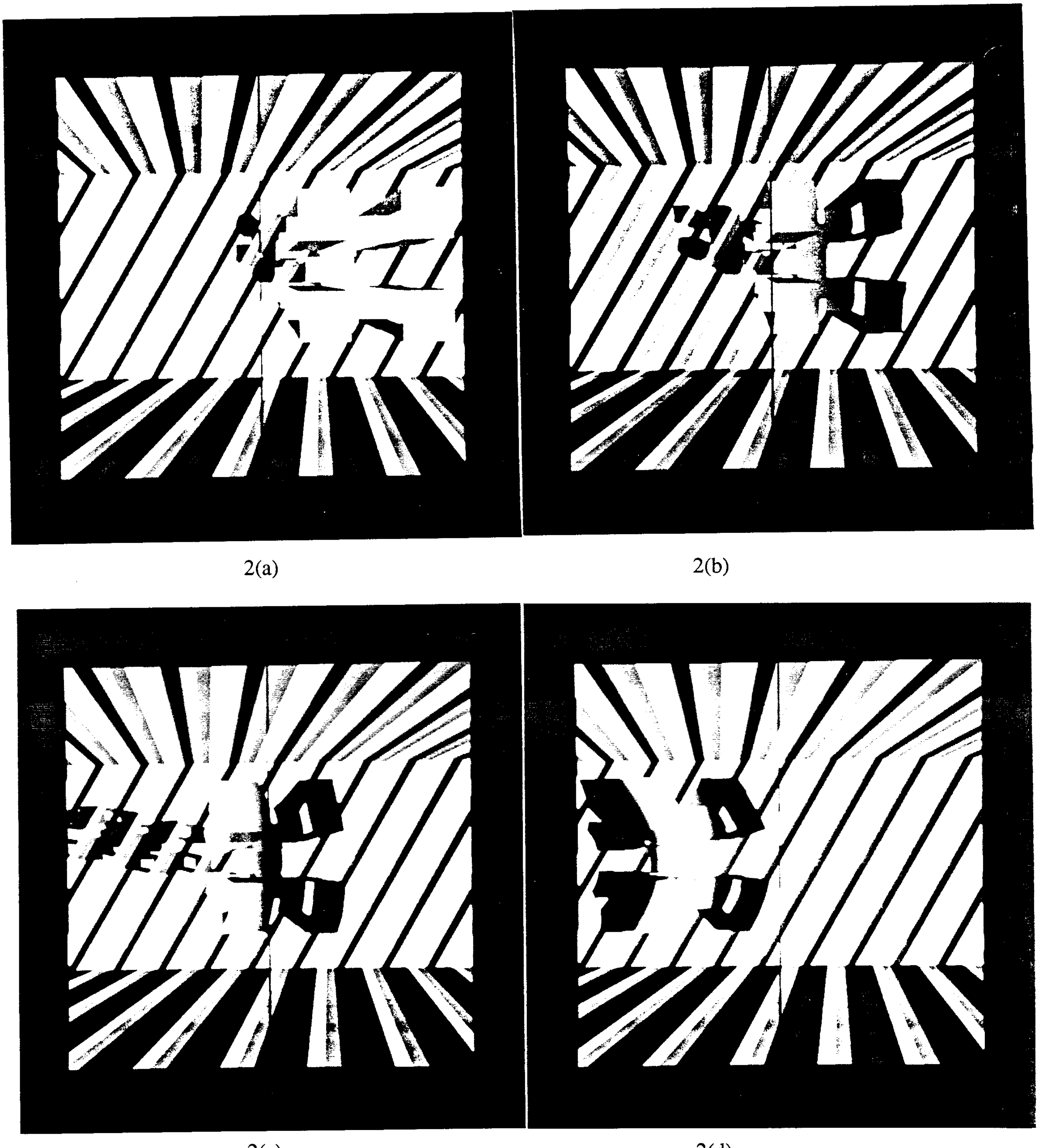


Figure 1. The rainbow resulting from our color mapping.
Here, the wavelength linearly increases from 380nm (far left) to 780nm (far right).



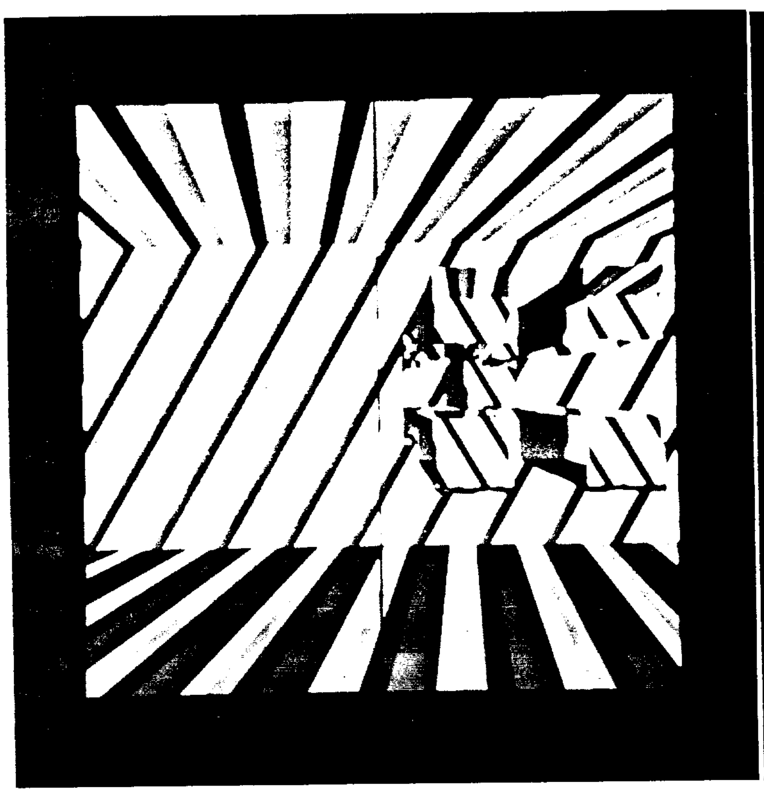
2(a)

2(b)

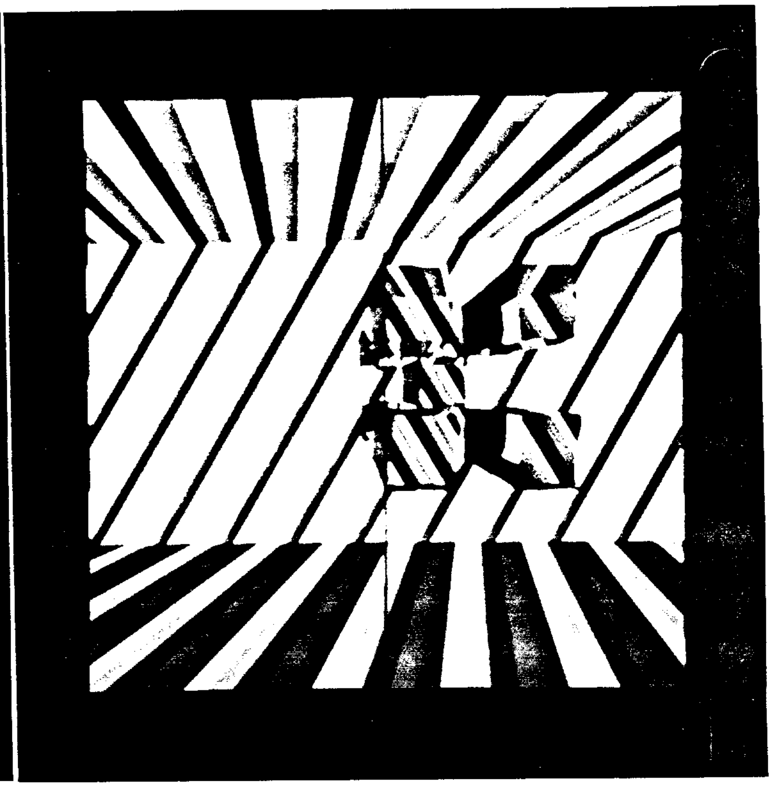
2(c)

2(d)

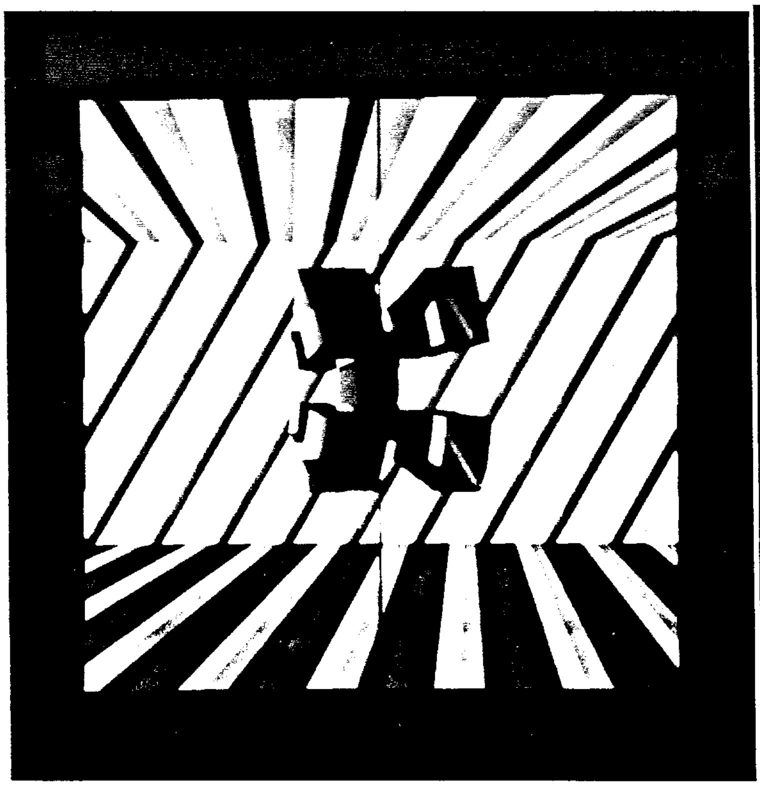
Figure 2. The resulting distortions for ships traveling at different speeds:
2(a) at rest , 2(b) at 0.3c , 2(c) at 0.6c , 2(d) at 0.9c .



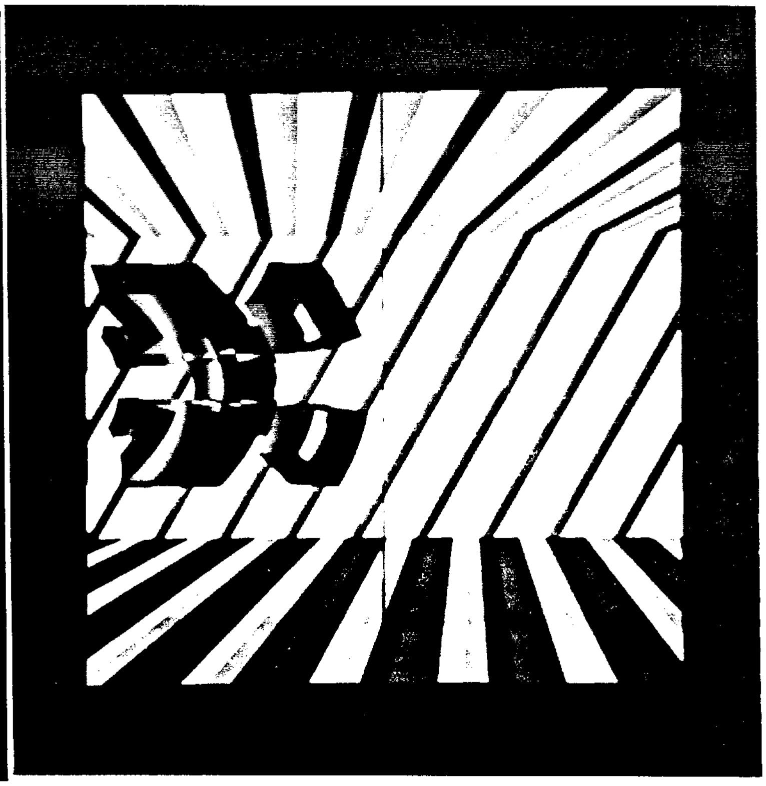
3(a)



3(b)

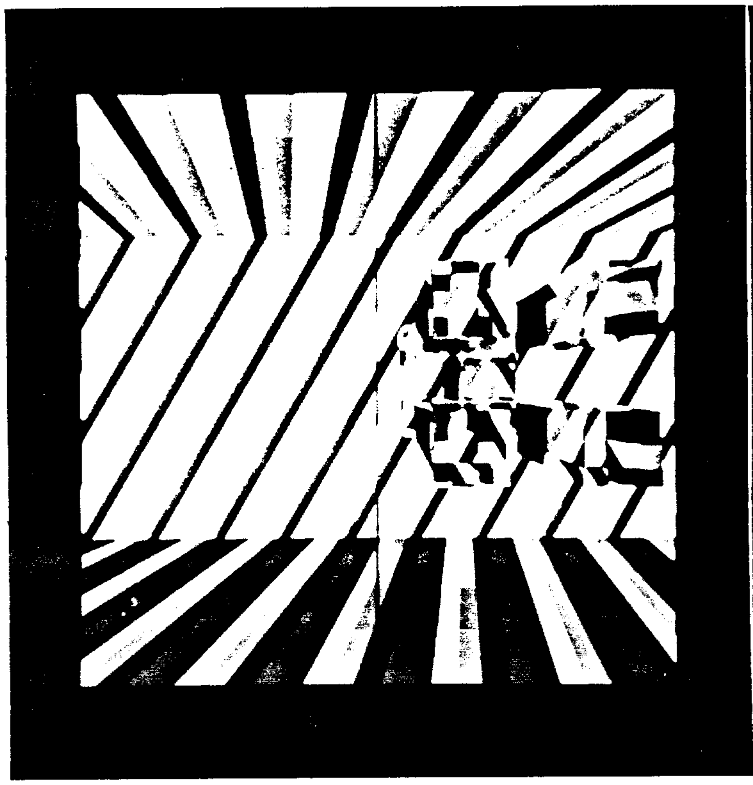


3(c)

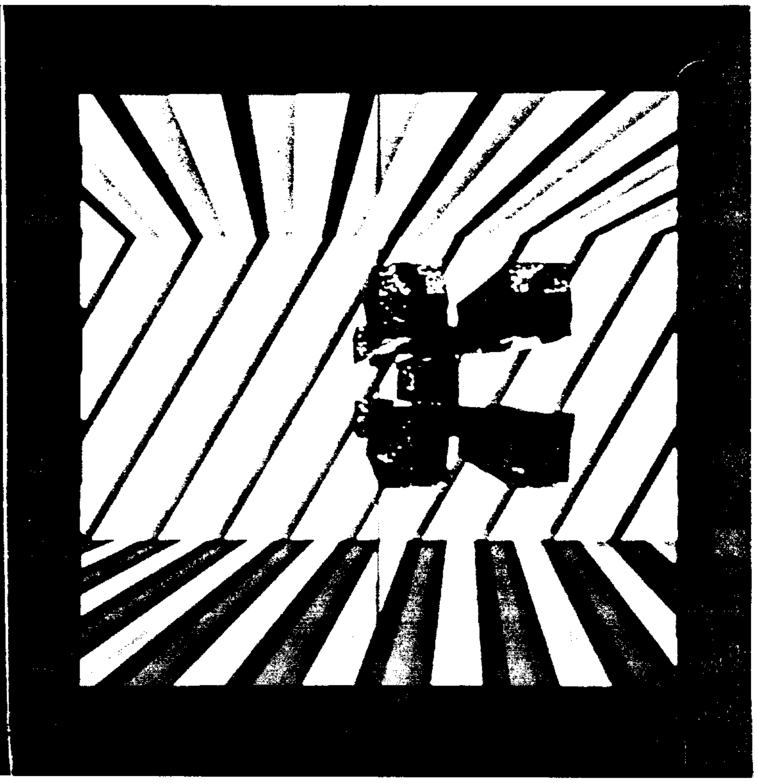


3(d)

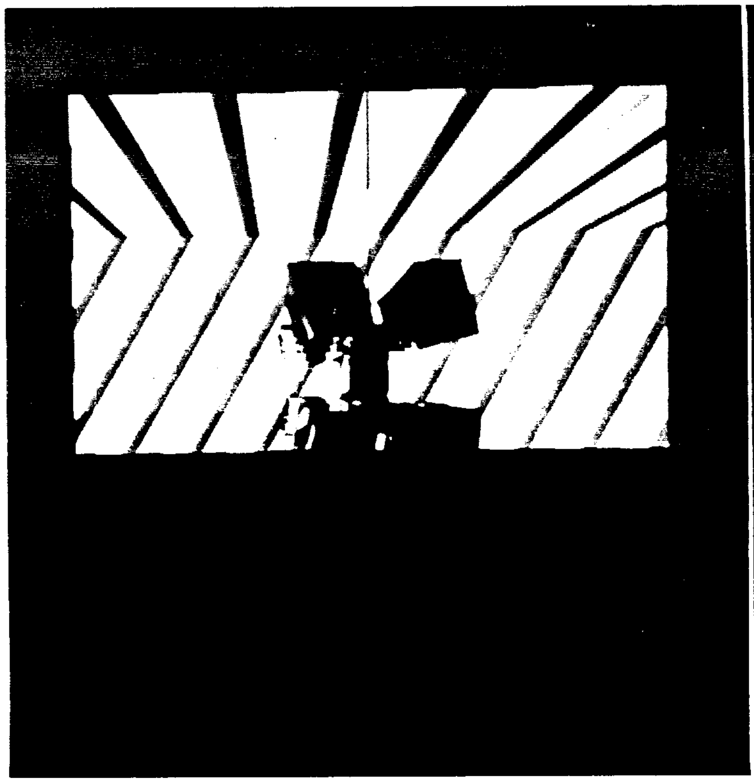
Figure 3. The same set-up as in Figure 2, but this time the images were rendered with reflections.



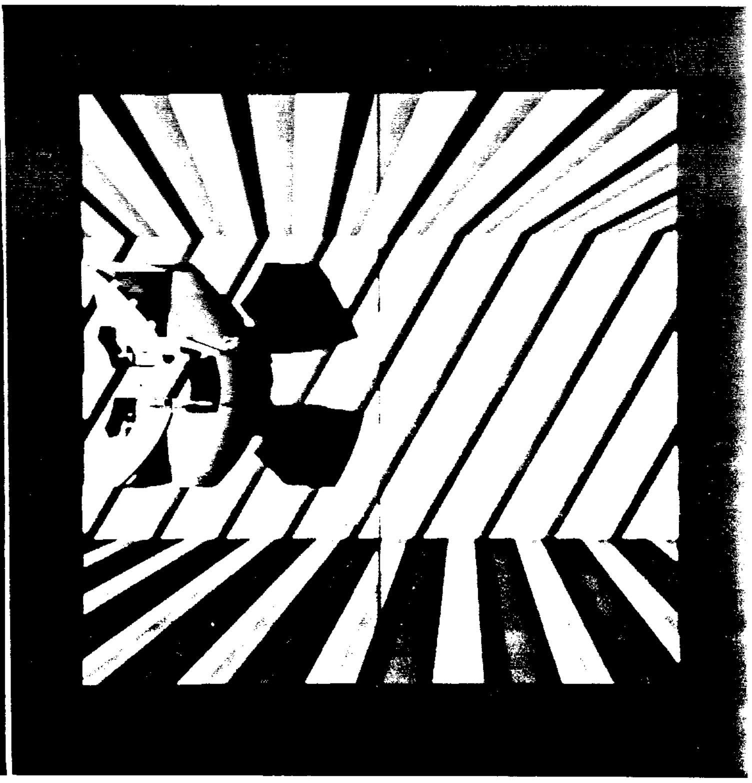
4(a)



4(b)



4(c)



4(d)

Figure 4. The same set-up as in Figure 2, but this time the images were rendered with refractions and reflections.