Real Time Relativity

Walter Gekelman, James Maggs, Lingyu Xu
Department of Physics
University of California
Los Angeles, 90024-1547

Abstract

A software package which calculates and displays in real time the shape of a cube moving at relativistic velocity in the sunlit world is described, and examples of its output are presented to illustrate the effects of high velocity on the appearance of a common object. The cube may be launched from any position and at any angle relative to the observer, but the velocity, $\beta = v/c$, is assumed constant. The parameters used in the program may be varied in real time using buttons and knobs. The entire program is menu driven, and one can chose to view the cube as a Doppler shifted object or to have each face colored differently to keep track of the large distortions which can occur. The paper contains the theory and computational method used to calculate and display the resulting shape. The most important subroutines are contained in an appendix.

Introduction

In introductory courses on special relativity students are taught about Lorentz contaction and time dilation with respect to two inertial frames in relative motion. Many students come away with the impression that if an object was moving toward them at an appreciable fraction of the speed of light, it would appear contracted in its direction of motion. It has been understood for many years that this is not the case, the Lorentz contraction applies in a world described by measurement with a lattice of clocks and meter sticks (Taylor and Wheeler, 1966). The "observation" of an object in this world rests in an analysis of data tapes issued by detectors and clocks within the lattice, long after the object is gone. All parts of an object must be measured at the same time in order to observe the phenomena of Lorentz contraction. The difference between a human observer, or a camera, and this type of measurement is that a light sensor, at any given instant of time, detects light which may have originated from the object at very different times. This effect was recognized by several authors (Weisskopf, 1960; Terrell, 1959; Penrose, 1959) nearly thirty years ago and several calculations were

done to find the shape of simple objects moving at relativistic speeds, as seen by a human observer.

For several reasons, little of this has filtered down to the classroom. Students generally have such a hard time with relativity that many instructors feel additional information may lead to an irreversible overload. Also, there has been very little available in visualization tools to dramaticially illustrate relativistic effects. A large part of the difficulty that students have with Physics is an inability to form a picture which captures the essence of the subject apart from the mathematics in which it is couched. If the problem involves three (or more) dimensional forms changing in space and time, ordinary blackboard diagrams become nearly useless.

Fortunately, the introduction of powerful graphics workstations is changing this picture. This paper describes, in detail, an interactive program which runs and renders, in real time, a cube moving at constant relativistic speeds in any direction with respect to an observer. The code can be straightforwardly modified to deal with any shape or any velocity trajectory. The paper is orgainized as follows. First we describe the problem in more detail and review some of the ways others have tackled it. We then present our algorithim for solving it, before proceeding with a description of the program and its user interface and a presentation of several examples of the output. Finally, we list the most important parts of the code.

The visual appearance of an object moving at relativistic velocities

Consider the emission of light from a simple object, namely a cube, moving rapidly toward an observer in the absense of gravitational fields. As shown in Figure 1, a spherical wave emitted, with the cube at rest, from a point P on the rear surface is blocked by the back of the cube, and is therefore not visible to the observer at point X. Diffractive effects are not considered here. Light rays are normal to the spherical surface, and can not bend around corners. In contrast , when the cube moves rapidly towards the observer at velocity v along x, it can outrace most of the expanding light sphere, and the ray emitted from the point P on the rear face can get to the observer's eye. The angle of elevation, Ψ, at which this happens is given simply by $\Psi_\beta = \cos^{-1}(v/c)$ (Taylor, 1963). It is therefore possible to see the rear side of a cube approaching at relativistic

speed, from a viewing angle at which it would not be if the cube were at rest!

If the cube is far enough away so that every point on it subtends approximately the same angle Ψ with respect to the observer, the cube will appear to be rotated as a solid object (Weisskopf, 1960). The cube appears to rotate as a solid object because it is Lorentz contracted. If it were not Lorentz contracted it would appear to be elongated along its direction of motion. The relationship between the angle of observation, Ψ, and the angle of apparent rotation, φ, is given by:

$$(1) \qquad \varphi = \arccos \left\{ \frac{\cos \psi - \beta}{1 - \beta \cos \psi} \right\} - \psi$$

When $\Psi = 0°$, the object moves directly towards the observer and the rear face is never observed (i.e., $\psi + \varphi = 0$). When Ψ increases for a fixed large β, ($\beta = v/c$) the rear face of the cube comes into view at the angle of elevation Ψ_β (i.e., $\psi + \varphi = \pi/2$). In the limiting case as $\beta \rightarrow 1$, the cube appears to rotate so that only the rear face is visible from any observation angle (i.e., $\psi + \varphi = \pi$). If one plots the angle of apparent rotation of the cube as a function of the angle of observation (Taylor, 1966), as shown in Figure 2 another interesting phenomena emerges. For large β ($\beta > .95$) and a certain range of observation angles ($\Psi \leq 90°$), the cube can appear to rotate more than 90 degrees. In these cases, the bottom face of the cube appears to swap places with the front face, and the rear face with the bottom face, as illustrated in Figure 3. For small β (0.5) the object rotates slightly ($\phi = 18°$). As β approaches unity the cube can rotate more than 90 degrees so that the rear is visible to the observer.

This simple analysis breaks down when the cube is close enough to the observer that each point on it subtends a significantly different observation angle Ψ. In these circumstances, one could as suggested by Taylor, approach the problem by breaking the cube up into a multiplicty of smaller cubes and then calculating and performing the above rotation on each cube. This conceptual proceedure becomes quite cumbersome in a calculation, since the algorithim must determine how many cubes to break the mother cube into, and then find a way to smoothly join the resulting bunch of differentially rotated daughter cubes for graphical presentation.

The problem is further componded if the original object is a smooth curved surface and not easily cubized. Because of these difficulties, we choose not to use this approach.

There are other aspects which must also be incorporated into a visual presentation. One of these is the Doppler effect which will change the wavelength of the light from the surface of the object according to the relation,

$$\lambda = \lambda_0 \gamma^{1/2} \{ \frac{1.0}{(1 + \beta \cos(\Psi))^{1/2}} \} \tag{2}$$

where $\gamma = (1-\beta^2)^{-1}$. As the cube approaches it becomes bluer, and is red shifted when it receeds. Another effect is relativistic magnification. One sees the rear of a rapidly approaching object in the quasi-remote past, and it will appear smaller than the front surface since it was further away when it emitted (or reflected) the light. Finally, there is the searchlight effect in which the distribution of light emitted from a rapidly moving object is most intense along the direction of motion. This effect occurs because, as seen from the observers viewpoint, the spherical surfaces containing emitted light energy are closest together along the objects direction of motion, and thus the light intensity is highest in this direction. The distribution of light intensity is given by (Weisskopf, 1962)

$$I(\theta) = I(\theta') \{ \frac{1 - \beta^2}{(1 + \beta \cos(\theta))^2} \} \tag{3}$$

Here θ is the angle of observation of the emitted light with respect to the surface of the object, and θ' is the angle of light emission in the frame of the object.

The problem is how to handle all these effects in a situation where real time interactivity is essential. The ray tracing method circumvents all the calculations involving rotations by simply following each ray from the eye of the observer back to the object, and keeping track of the different propagation times. This technique has been used by Peterson (1990) in an article which contains many striking visual displays. Ray tracing, however is a time consuming process which can take tens of minutes to hours to generate one image. At this time there is no hardware which can ray trace in real time, so we decided to

develop an algorithim based on ray tracing concepts which could be implemented on one of the new breed of supergraphics workstations.[1]

Calculation of the Appearance of a Relativistically Moving Cube:

Light, emitted or reflected from a moving object, reaching an observer's position at time t=0, travels various paths of differing length. In order to reach an observer at the same instant, light from a section of the object farthest away from the observer must be emitted earlier than light from the nearest section of the object. Moreover, since the object is in motion, it is in a different position at an earlier time, so that a snap shot of the object in relatvistic motion could be distorted from its shape at rest.

Suppose we know the rest shape of an object, and represent it in the computer as an array of points on the surface of the object. In order to compute the spatial location of various points on the object's surface when they emit rays reaching the observer at t=0, it is convenient to consider the spatial location of the object at the time the ray traveling the shortest path length reaches the observer. This ray is emitted from the point on the object nearest the observer. The light emitted from the nearest point travels a distance r_n (r_nearest) to reach the observer, in a time interval of length r_n/c . All other points on the object's surface lie outside a sphere of radius r_n centered about the observer, as illustrated in Figure 4. Denoting the position vector of a point on the object's surface, measured from the observer's location by $\mathbf{r}(t)$, (vectors are denoted by bold face) with $\mathbf{v}(t)$ the velocity vector, the object is in position $\mathbf{r}(t=-r_n/c)$ moving at velocity $\mathbf{v}(t=-r_n/c)$ when the ray from the nearest point reaches the observer. The trajectory of the cube need not be a straight line moving at constant velocity. For simplicity of analysis, however, we will consider the velocity constant, i.e., $\mathbf{v}(t) = \mathbf{v}(t=0)$. The extention of the method to an accelerated trajectory will be discussed after analyzing the constant velocity case.

Now we calculate the position of some point located on the object's surface (but not the nearest point) when it emits a light pulse that arrives at the observer at t=0. The spatial location of this point at time $t=-r_n/c$ is $\mathbf{r} = \Delta\mathbf{r} + \mathbf{r}_n$, where

$$\Delta\mathbf{r} = \Delta\mathbf{r}_0 - (\Delta\mathbf{r}_0 \cdot \boldsymbol{\beta})(1 - \gamma)/\beta \qquad (4)$$

[1] In this case a Stardent Titan (64 Meg memory, 2 CPU's)

The vector Δr_0 is the displacement vector from the nearest point to the emitting point measured when the object is at rest. Notice that the expression (4) contains a Lorentz contraction factor in the direction of the particle velocity, because the location r is determined relative to r_n, the location of the nearest point, at a fixed time, namely $t=-r_n/c$. In order for a pulse of light emitted from the point located at r to reach the observer at time $t=0$ it must penetrate the spherical surface of radius r_n about the observer's postion at the time $t=-r_n/c$. If the position of the point at the time the pulse is emitted is denoted by r_{ret}, then the time of flight along the ray path before the pulse penetrates the sphere of radius r_n, is, $t = (r_{ret} - r_n)/c$ and the emitting point on the object has moved a distance

(5) $$d = vt = \beta(r_{ret} - r_n)/c$$

from its location.

The location of the point at the time of emission, r_{ret}, is related to, r, by

(6) $$d = r - r_{ret}$$

where d is given by (5). Taking the vector dot product of both sides of (6) and using (5) gives the expression

(7) $$\beta^2(r_{ret} - r_n)^2 = r^2 + r_{ret}^2 - 2r \cdot r_{ret}$$

Taking the vector dot product of (6) with r and using (5) to replace d, the resulting value for $r \cdot r_{ret}$ used in (7) gives

(8) $$r_{ret}^2 (1-\beta^2) - 2r_{ret}(r\cdot \beta - \beta^2 r_n) = r^2 - 2r\cdot \beta\, r_n + \beta^2 r_n^2$$

Equation (8) can be solved for r_{ret} using the standard solution for quadratic equations,

(9) $$r_{ret} = \frac{1}{2}\{ -b_1 + (b_1^2 - 4a_1c_1)^{1/2} \}$$

where:

(9.a) $$a_1 = 1-\beta^2$$

(9.b) $$b_1 = 2(r\cdot \beta - \beta^2 r_n)$$

(9.c) $$c_1 = -(r^2 - 2r\cdot \beta\, r_n + \beta^2 r_n^2)$$

The set of all end points of the spatial location vectors, r_{ret}, of points on the surface emitting pulses that arrive at the observer at the same instant of time (t=0, for the case under consideration) is called the *photosurface*. The photosurface is generated by the program from the object's location at $t=-r_n/c$, using equations (4) - (9). The photosurface can be selected for rendering and viewing from any angle. Of course ,the appearence of the object is found only by viewing the photosurface from the observers location. This view of the photosurface is how the object would appear if a camera located at the observers position took a snap shot of the object at the time t=0.

Another way to represent the relativistically moving object is to transform it so that its appearance, when viewed by the observer, is the same as the appearance of the photosurface. We first rotate each point on the surface of the cube using the expression for the apparent rotation angle of a cube of negligible size (given by equation 1) located at the retarded position r_{ret}. The elevation angle of the point located at r_{ret} is given by

$$\psi_{ret} = \arccos (r_{ret} \cdot \beta / \beta r_{ret}) \qquad (10)$$

As illustrated in Figure 5, the rotation of this particular vector (delta_r) occurs about an axis in the direction of $r \times \beta$ (which is the same as $r_{ret} \times \beta$). This axis of rotation has been named **rotme**. Once the point on the object's surface is rotated in this fashion, the vector from the observer to the rotated point ($r_{rotated}$ in Figure 5) is projected onto the direction of the corresponding point on the photosurface. This process ensures that each point on the transformed cube is along the observer's line of sight to the corresponding point on the photosurface. The rotated-projected cube will then appear identical to the photosurface from the observers viewpoint. The transformed cube is generated by the program using equations (1) and (10), and can be selected for viewing from any angle.

Both the photosurface and rotated-projected cube appear identical only when viewed from the observer's position. However, both objects can be viewed on the computer screen from positions other than the observer's position. Usually their appearence is strickingly different. This ability to view the objects from various aspects can be thought of as a second observer observing the original

observer and cube. The second observer's view of the photosurface and transformed cube can not be realized in the physical world, but provides some instructive insights into the appearance of the relativistically moving cube.

The procedures used to find the photosurface and transformed cube can be easily generalized to objects of more complex shape and accelerated trajectories. The shape of the object is a problem only as regards program speed. The data input required is an array of points on the surface of the object at rest. The complexity of the object's shape, or accuracy of its description, is then limited by the array size. Too large an array will slow the program to the point where it can no longer be considered interactive.

An accelerated trajectory can be handled by replacing equation (5) with the expression

$$(11) \qquad \mathbf{d} = \int_{t_n - t}^{t_n} dt'\, \mathbf{v}(t') \equiv \bar{\mathbf{v}} t,$$

with $t = (r_{ret} - r_n)/c$, and where $\bar{v}$ is the average velocity over the interval from t_n-t to t_n. The vector r_{ret} can then be found using an iterative approach. The average velocity is first approximated by setting it equal to $v(t_n)$, that is, its instantaneous value when the ray from the nearest point is emitted. The vector r_{ret} is then found as in the constant velocity case, and the time interval used in equation (11) is found using $t=(r_{ret} - r_n)/c$. This new value of the average velocity is then used, and the procedure repeated until the change in the average velocity after the iteration is below some preset criteria (e.g. $|\Delta \bar{v}/ \bar{v}| < .01$).

The velocity trajectory is followed by breaking it up into segments, along which the velocity is constant. At one time step the shapes of the surfaces are computed as described above. At the next time step in which $t_n \rightarrow t_n + \Delta t(n)$, the spatial location of the cube is advanced using velocity v_n, and the new surfaces are computed as before. Clearly the repeated procedure for finding the average velocity corresponding to each point on the surface could greatly slow the program. It can be speeded up by replacing the first estimate of the average velocity by the value found at the previous time step for the point in question, or by some similar procedure

depending upon the technique used to evaluate the integral in equation (11). For example, if an extended trapezodial rule is used to numerically evaluate $\bar{v}$ that is,

$$\bar{v} = \frac{1}{M}\left\{ \sum_{m=0}^{M} [v(t_n - m\tau)] \; -(v(t_n - t) + v(t_n))/2 \right\}, \text{ where } \tau = t/M,$$

the integration can be updated at each time step by adding the term $[v(t_n+\Delta t(n))+v(t_n)-v(t_n-t+\tau)-v(t_n-t)]/2M$. If the acceleration along the trajectory is not large this value will be close to the correct expression.

In addition, the magnitutde of $\Delta t(n)$ need not be the same for each time step. The value of $\Delta t(n)$ can be determined, for example, by limiting the size of the derivitive of the velocity at each step. That is, requiring $|v(t+\Delta t) - v(t)|/|v(t)| < \varepsilon$, where ε is a small, arbitrarily choosen, positive number. In this case the value of $\Delta t(n)$ varies for each time step, and can adequately represent the motion when the acceleration is large.

Running the Realtiview Program

The user interface for the relativity program is structured so that only a mouse and a dial box are used. Once the program is initiated by typing RUNME from the control console, a main window and several border windows appear as shown in Figure 6. The biggest window which is positioned on the upper left hand part of the screen is the Dore window. Dore (Dynamic Object Rendering Enviornment) is an object oriented software [2] graphics system All of the objects required such as the cube and the "gun" which fires it are rendered within the Dore window. The cube at rest is shown with the bottom face colored white and the face facing the observer colored magenta. A three dimensional grid centered at (0,0,0) is displayed. The rectangular coordinates range from -100 $\leq$ to $\leq$ 100 ls (light seconds, or the distance light travels in a second. Each time interval is a second). At the bottom left of the screen there is an explanation/instruction window which contains a brief abstract of the program. To the right of this there is an I/O (input-output) window which is initially blank. It has been used during program development for debugging purposes

and can show the instantaneous value of a parameter of interest such as Ψ, the average elevation angle in real time.

A stop sign, displayed on the lower right of the screen is used to exit from the program. To exit, the mouse is positioned within it and clicked.

A button window is located in the upper right corner of the screen. To "press" a button the mouse arrow is positioned on one and clicked to activate it. The button commands are explained in detail below.

A window on which a set of dials, very much like those on the dial box, is drawn on the lower right corner of the screen. There is a one to one coorespondance between the dial icons and the physical dial set. The function of each dial is written on the screen below it. If the dial hardware is not present, a dial may be activated by placing the mouse on it and clicking . A button labelled "knob set A" can toggle between alternate knob sets since there were more knob functions necessary to run the program than physical knobs. By clicking it one can toggle between knob sets A and B.

The buttons displayed on the screen perform the following functions:

1. Rep Type : This button sets the mode in which solid objects are drawn. Objects can be displayed as points, in wireframe mode, or as a shaded surface.
2. Shading : This button sets the shading type. The cube and gun can be flat or Gourard shaded.
3. Highlights: This button determines whether glossy highlights will be present or not
4. Time steps : This button sets the maximum number of time steps for the animation which can be any positive integer number. The default number is 100. If an object moves slowly it may not go far in 100 steps.
5 Background: This the screen background color to black, red, green or blue.
6 Box color:. This button can set the color of the cube to either a single color or a seperate color for each face. The single color, which is set to green when $\beta = 0$, is used to illustrate the Doppler effect. As $\beta \rightarrow 1$ the cube becomes blue and turns red for $\beta \rightarrow -1$. The seperately colored surfaces are not Doppler shifted but are useful when studying the

rotations and extreme distortions of the cube at large β and arbitrary direction of motion with respect to the observer.
7 Object. This button determines the viewing case. In one instance, the object displayed on screen is the photo-surface, and in the second instance it is the rotated-deformed cube.
8. Camera: This button switches the camera between two positions. In one case the camera is positioned at the observer's location. In the second case the camera may be positioned anywhere, and the observers position is denoted by an X along the x axis. This case corresponds to an observer observing the original observer.
9. Animate : This button starts or stops a clock which determines the time intervals between succsive positions of the cube. When the cube is moving and the animate button is pressed, the cube will remain frozen at its last position
Next we explain the function of the knobs. The knob box contains eight knobs which, unfortunately, are not enough to handle all the functions required by the program. To overcome this difficulty there is a button beneath the knob descriptor (Figure 6) which, when clicked, changes the function of the knobs. A short description of its function is written underneath each knob.

Knob Set A:

All the dials in this set are used to set the cube's initial position and velocity in 3D space. A gun is shown with its muzzle pointing in the direction of the cube velocity.
Dials 1,2, and 3. These dials set the cube's initial x, y and z position on the rectangular coordinate axis.
Dial 4: This dial sets the observer's position on the x axis. The current observer's location is displayed by the symbol X
Dial 5,6: These dials set the spherical coordinate angles theta and phi. The angles are in degrees and written below the knob as it turns.
Dial 7: This dial sets the magnitude of the cube's initial velocity.
Dial 8: This dial changes the intensity of the lights which illuminate the cube.

Knob set B:

All the dials in this set are used to control the orientation of the volume viewed with respect to the second observer. They do not change any of the parameters of the cube motion.
Dials: 1,2,3: These dials rotate the entire grid around the x, y, or z coordinate axis. The degrees rotated are shown below the

corresponding icon and may take on positive or negative values. Positive rotations are determined by the right-hand rule.
Dial 4: This dial zooms the camera in or out from the grid origin
Dials 5,6: These dials translate the entire grid relative to the center of the screen in the x and y directions.

Examples

Figure 7a shows the appearance of a cube traveling at $\beta = .99$ in the x direction towards an observer located at x = 600 ls. The cube is intially located at (x,y,z) = (0,80,0) so that its elevation angle is 7.5 degrees. Figure 7b shows the photosurface for the same case as 7a. from the perspective of an observer looking at both the cube and the true observer. Note the photosurface is highly elongated since β is large, but as illustrated in Figure 7c, both the cube and photosurface appear identical from the perspective of the observer located at x=600. Even though β is large, the apparant rotation (Figure 3) is less than ninety degrees since the angle of elevation is small, and the observer sees the (white) bottom and (magenta) front surfaces of the cube. Figure 8 shows the distortion of a large size cube due to variation of elevation angle from bottom to top. The cube center is at y = 20 ls while its lower edge is at y = 2 ls, and the observer is much closer at x = 100 ls.. The top edge of the cube subtends an angle of 11.3 degrees, and with $\beta = 0.95$ a small cube would appear to rotate by about 55 degrees. The bottom edge subtends an angle of only 1.1 degrees so a small cube would only rotate about 5 degrees. The large cube in this figure differentially rotates and thus its front surface becomes rounded.

In the next sequence, a cube centered at the origin and moving along the x-axis with $\beta = 0.95$ approaches an observer at x = 20 ls. Before the cube reaches the observer (Figure 9a) it is blue shifted and appears elongated, nearly bullet shaped. Light from the back of the cube reaches the observer from the past when the cube was much further away so that the rear is squashed down. In Figure 9b the cube is on top of the observer and its overall color shifts toward green which is the object's color at $\beta = 0$. The leading edge appears large, since it is beyond, but close to the observer so that it subtends a large solid angle. Finally the cube turns red and appears more cubelike (Figure 9c) as it receeds from the observer. Since it was difficult to correctly color each part of the cube according to its elevation angle (equation 2) an average color was used in this demonstration. Finally,

for the same β and observer position as in the previous case, the cube can assume very un-cubelike appearances when it is launched at an arbitrary angle such as $\theta = 112.5^{o}$ and $\phi = 43.5^{o}$ as shown in Figure 10. The velocity direction angles are the standard angles used in sperical coordinates, with θ measured from the positive z-axis and ϕ measured from the positive x-axis.

Program Structure

The code can be roughly grouped into three classes of software. The main part, and the focus of most of this paper, is the subroutine which calculates the positions of points on the surface of the relativistic object as a function of time. This module, *realtiview*, is written in FORTRAN and is described in detail below. The next module, *geom_spec* , is written in C and uses the data passed from *realtiview* to create the graphical objects. (In this case the cube and gun which fires it) One the first things that Geom_spec does is pass the data on the object's surface points to a routine called *PATCHFIT.C* (Green, 1990) which takes an array in rectangular coordinates and fits a set of triangular patches to it. Any graphical object is constructed out of a number of these patches. As the object becomes more deformed this routine may create tens of thousands of patches to accurately fit it. This means that highly deformed cubes, which are moving close to the speed of light, wind up moving more slowly across the screen since the computation time goes up as the number of patches. *Geom_spec* next calls the Dore library which in turn renders and colors the objects in three dimensional space. The main graphical program *Main.C* is not unlike the conductor of an orchestra. It initializes the X windows, controls all the peripherals such as the mouse and knobs, and allows the user to interact with the objects. These latter routines are complex, took many man years to write and, fortunately can be ignored for the most part. They are simply linked into the program. The total number of modules in this package is 22, and requires access to nine libraries (i.e. FORTRAN77 ,C, X11, mathlib, Dore, etc.)

Realtiview

The FORTRAN subroutine which does all the calculations is entitled *realtiview* (Realtime relativity view). It was written in FORTRAN since this compilier generates faster and more effeciently

vectorized code. A commented listing of it is provided as an appendix . It calculates and passes the points on the photosurface and transformational cube to the graphics programs. The first array, surface, is the rotated and deformed cube as seen by an observer at location **r** = (X,0,0). The second array, ph_surface, is the photosurface. Other information about the observer's position, and gun location is passed to this routine as well. For every time step, 'it', *realtiview* evaluates the position, shape and color of the cubes, and passes them to the graphics package for rendering.

In the first time step, it= 0, *realtiview* creates two cubes. One cube (**surface**) is sized normally and is used to calculate the transformed cube, and the other (**Lorentz_surf**) is Lorentz contracted, and used to calculate points on the photosurface. The transformed cube is placed at the origin and rotated to line up with the gun. All rotations are done point by point using the standard transformation and rotation matricies (Harrington,1987). In general every time this subroutine is called an array of vectors, **r_surf**, from each point on the Lorentz surface to the eye of the observer is created, and an axis of rotation for each surface point is evaluated from **r_surf**× **v** and placed in **rotme.**. Next the point on **Lorentz_surf** nearest to the observer's eye is found and stored in **nearest**. Then the photsurface is calculated from the current position of **Lorentz_surf**, according to **r** = **v**t. In addition, the angle of elevation of the points on the photsurface are found and stored in sigh_tot. The program then creates the distorted cube by rotating each point on **surface** by angle φ calculated from formula (1), about the axis in **rotme** The rotated surface is then moved so that the point on the surface corresponding to the nearest point is at position **nearest**.

Acknowledgements

This work would not have been possible without a grant from the UCLA Office of Instructional Development. The authors of this paper would like to thank the Stardent computer corporation for help in supplying needed software and for providing help with their graphics program Dore. We would also like to acknowledge many useful discussions with Prof. Joe Rudnick of the Physics department.

Figure Captions

1. Illustration of how light emitted from the rear of a rapidly moving cube can reach an observer a) Light emmited from a point on the back of a stationary cube is blocked by the cube. The hemispherical blue surface represents a light pulse emitted from the rear of the cube b) A rapidly moving cube emits a light pulse at the same time as the stationary cube in a). The cube races to the right and light from the expanding spherical wavefront is no longer blocked and can arrive at the observer.

2. Graph of equation 1, the apparent rotation of a small cube as a function of elevation angle and β. For $\beta > 0.95$ and a range of elevation angles the cube can rotate more than 90^o so that an observer will see the rear face.

3. Illustration of the rotation of a cube which subtends a small solid angle when it moves at intermediate and high β a) $\Psi = 30^o$, $\beta = 0.50$, $\varphi = 18.^o$ b) $\Psi = 30^o$, $\beta = 0.99$ and the rear face (colored blue) and the bottom (colored orange) are seen. The rotation angle $\varphi = 120^o$.

4. Some of the vectors used in calculation of point by point rotation of surface elements of the cube. The coordinate origin is at (x,y,z)=(0,0,0), the vector **eye** goes from the origin to the observer's eye. $\mathbf{r_n}$ is from the observer's eye to the nearest point on the surface of the cube, while **r** is a vector to an arbitrary point on the cube. $\mathbf{r_{ret}}$ is the location of the point on the surface (corresponding to **r**) when it emits a light pulse that reaches the observer at the same time as the light from $\mathbf{r_n}$. The point moves a distance **d**=**v**t in the time it takes the light emitted at $\mathbf{r_{ret}}$ to penetrate a sphere of radius $\mathbf{r_n}$ (drawn in blue) surrounding the observer.

5. Elements in the proceedure used to transform the rotated cube so that appears, to the observer, identical to the photosurface. A vector on the surface $\mathbf{r_{initial}}$ is rotated about the axis **rotme** by an angle prescribed by equation 1. The rotated vector, $\mathbf{r_{rotated}}$, is then projected onto $\mathbf{r_{ret}}$, the line of sight to the corresponding point on the photosurface. **Delta_r** is the difference between $\mathbf{r_{initial}}$ and **nearest**, the vector from the observer to the nearest point on the cube ($\mathbf{r_n}$).

6. The computer terminal showing thescreen layout used in the program. A cube is shown at rest at the origin from the perspective of the second observer.

7. A view of the transformed cube and photosurface from the perspective of the second observer, and both from the observer's viewpoint. The center of object at y = 80.0 ls, the viewer is at x = 600.0 ls, and the cube is moving along the x-axis with $\beta = 0.99$. This and subsequent figures are discussed in more detail in the 'Examples' section of the text.

8. A cube moving along the x-axis with $\beta = 0.95$ and center at 20.0 ls shows the differential rotation due to the changing viewing angle across the surface of a large object. The observer is at x = 100.0 ls.

9. A sequence showing the cube moving along the x-axis through the observers position at at x = 20.0 ls with $\beta = 0.95$, illustrating both the dramatic change in the appearance of the object and the doppler shift.

10. The cube can appear highly distorted for certain viewing aspects and high velocities, as illustrated here by a cube with velocity direction angles $\theta = 112.5°$, $\phi = 43.5°$ and $\beta = .95$.

References

E.F. Taylor,J.A. Wheeler, *Spacetime Physics*, pp17-22,W.H. Freeman Company, New York (1966)

Taylor, E.F.,*Introductory Mecahnics*, pp349-358,J. Wiley & Sons,New York (1966)

V.F. Weisskopf,Physics Today,pp24-27(Sept 1960)

J. Terrell, Phys. Rev.,116,1041, (1959)

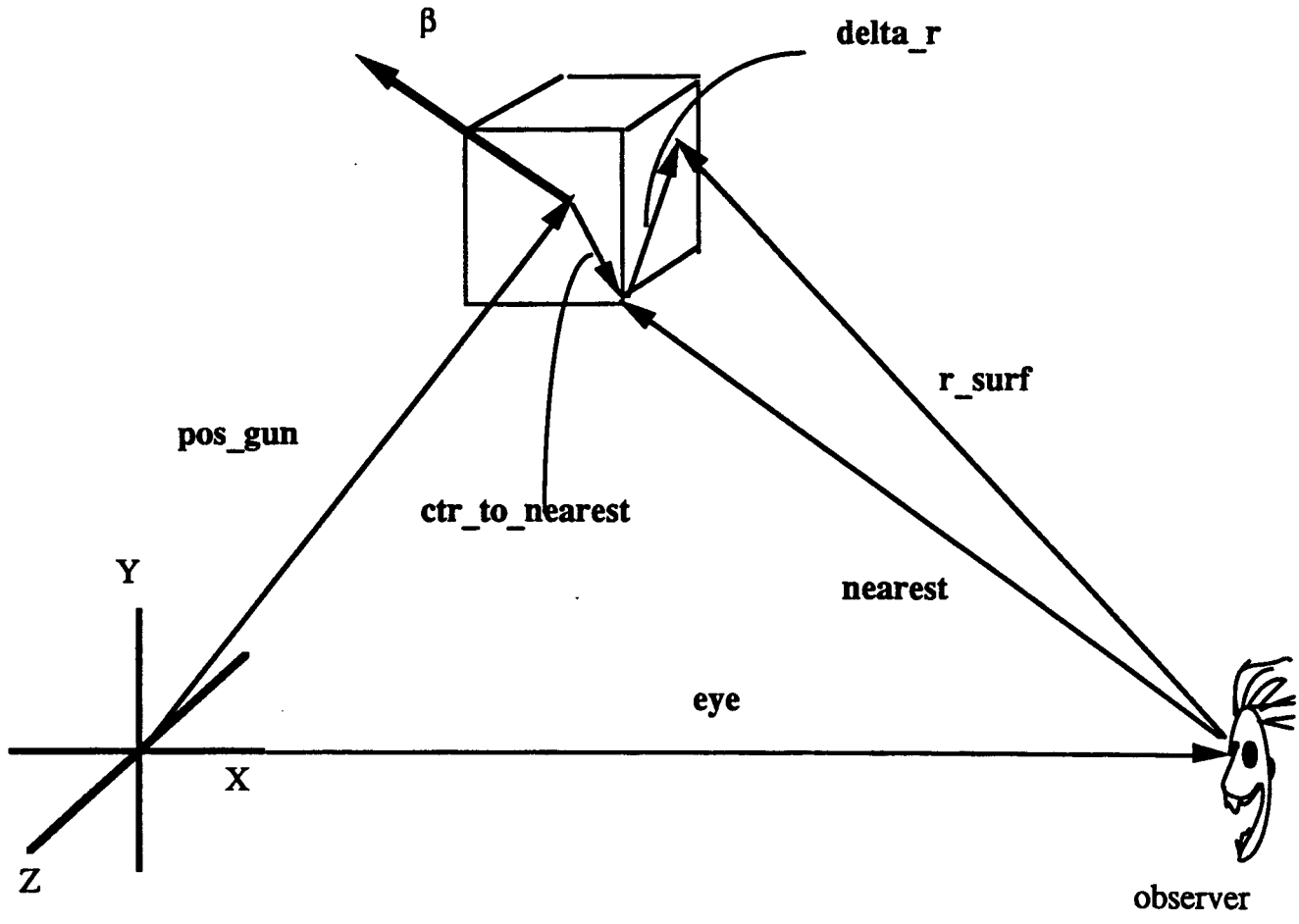
R. Penrose, Proc. Cambridge Phil. Soc. 55,137 (1959)

I. Peterson,SCIENCE NEWS,137,232(1990)

S. Harrington,*Computer Graphics, a Programming Approach*, pp256-261,McGraw Hill,New York(1987)

Appendix

```
      Subroutine  realtiview(surface,ph_surface,eye,orig,
1                                pos_gun,v,vvv,it,hue)
c
```




```
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     Input  parameters:
c     eye(3)          position of the observer's eye
c     orig(3)         position of the origin
c     v(3)            velocity of cube  0≤v≤.995
c     pos_gun(3)      position of gun which fires the cube
c     vvv(3)          θ, φ, v  angles gun makes and mag of velocity
c     it              time step ≥ 0
c     Output  parameters:
c     surface         surface of distorted cube
c     ph_surface      photosurface of cube
c     hue             Doppler shifted color of cube
c
c     The horizon is at y = 0
c     Written   W. Gekelman, J. Maggs, L. Xu 1989
```

```
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
c     Declaration of variables:
c
      real eye(3),orig(3)                    ! students eye position
                        !  and origin of cooorindate system position
      real pos_gun(3)                        ! init position of cubes
c                                            ! center
      real  eye_mag                          ! magnitude of eye
      real v(3)                              ! cube's center velocity
      real vvv(3)                            ! theta , phi , v_mag
      real t,temp                            ! time
      real  theta                            ! angle as seen by observer
      real  del_theta                        ! angle of rotation of each
      real  beta,c                           !  v/c
      real  theta_prime,time                 ! angle seen by observer
      real  pos_old(3),pos_new(3)            ! original position of cube
      real  hue,hue0                         ! color 0.0-0.5
      real  color_norm                       ! normalization for color
      real  angle              ! average angle cube makes with horizon
      real  gamma                            ! relativistic factor
      real  bottom                           ! used in Doppler calc
      real  r_ret_hat(6,16,3)                ! unit vector in r_ret direc.
      real  surface(6,16,3)                  ! points on object's surface
      real  ph_surface(6,16,3)               !  photosurface
      real  surface_orig(6,16,3)             ! created surface
      real  Lorentz_surf_orig(6,16,3)        ! surface with contractions
      real  Lorentz_surf(6,16,3)             ! surface with contractions
      real  r_surf(6,16,4)                   ! store scratch rpoint,magr
      real  rotme(6,16,3)                    ! rotation axis
      real  sigh_tot(6,16)                   ! total rot angle each point
      real  nearest(4)                       ! nearest point to eye
      integer it,i                           ! time step
      save  surface_orig,Lorentz_surf_orig      ! keep on reentering
c
      data  color_norm/1.414214/
c
c           ****************************************
c
c     The original position of the cube is stored in the
c          1st of 16 arrays
c
```

```
c          1st of 16 arrays
c
      hue0 = 0.25                    ! 0=red,0.25=green,.50=blue
      c = 1.0                        ! normalized speed of light
      if (it.eq.0)then               !  is it the first pass?
       beta = 0.0
       do i = 1,3                    ! save original position
c                                    ! of the gun
        pos_old(i) = pos_gun(i)          ! initialize position
        pos_new(i) = pos_gun(i)          ! ditto
        beta = v(i)**2 + beta
       enddo
       beta = sqrt(beta)/c
       if(beta.ge.1.0)beta = 0.99999     ! protection on divide
       gamma = sqrt(1.0-beta*beta)    ! definition of relativistic factor
c
c     Calculate position of points on surface given pos_new and beta
c     First create the cube with center at the orgin
c
        call create(surface,Lorentz_surf,beta,gamma)       ! create cube
c
c     Rotate cube so that face 6 is in beta direction, and place it at
c          initial position pos_new    position of cube center
c
        call rotate(Lorentz_surf,v,beta)          !Lorentz surface
        call rotate(surface,v,beta)     ! rotate uncontracted cube
c
c     Next:  point cube in beta direc. and translate to ipos()
c

      do isur = 1,6                  ! save newly created surfaces
       do ipnt = 1,16                    ! 16 points/side of cube
        do ix = 1,3                      ! x , y z
          surface_orig(isur,ipnt,ix) = surface(isur,ipnt,ix)
          Lorentz_surf_orig(isur,ipnt,ix) = Lorentz_surf(isur,ipnt,ix)
        enddo
       enddo
      enddo
c
      endif                          ! it = 0  surfaces done
c
      time = it                      ! make it real
      do i = 1,3
```

```
      pos_new(i) = pos_old(i) + v(i)*time
c                                ! for visible motion on screen
      pos_gun(i) = pos_new(i)    ! pass position of cube center
     enddo
c
c          Move surface to next position Dist = vel*time
c
     do isur = 1,6      ! new position of undistorted but Contracted
c                       !      surface
      do ipnt = 1,16
       do ix = 1,3
        Lorentz_surf(isur,ipnt,ix) = Lorentz_surf_orig(isur,ipnt,ix)
     1                          + pos_new(ix)          ! move em out
        surface(isur,ipnt,ix) = surface_orig(isur,ipnt,ix)   ! rawhide!
       enddo
      enddo
     enddo
c
c    Find the position vectors of points on the surface relative
c     to the observer - store them in r_surf.
c
     call  calc_r(Lorentz_surf,r_surf,beta,eye)
c
c    Calculate axis of rotation for each point on the surface.
c
     call  axis_of_rot(r_surf,v,beta,rotme)
c
c    Find nearest point to observers's eye and keeps track of it
c    nearest is the point on cube through which overall rotation will
c    be done,  ie rotation will be done about axis in rotme direction
c    and point of rotation at nearest!
c
     call nearest_pt(r_surf,nearest,eye,eye_mag)   ! find nearest
c                    !      point on surface to the observer
c
c    Calculate the photosurface - store results in ph_surface
c    Calculate elevation angles at retarded position - store
c    values in sigh_tot.
c
```

```
c     relative rotates each point on the cube as if it was a
c     microscopic cube.
c
      call  relative(surface,nearest,sigh_tot,r_surf,rotme,
     1            r_ret_hat,eye,v,beta,pos_new,angle)
c
c      Relativistic rotation based upon elevation angle of the
c       retarded position
c
      do isur = 1,6      ! new position of distorted surface
       do ipnt = 1,16
        do i = 1,3
          surface(isur,ipnt,i) = surface(isur,ipnt,i)
     1                         + pos_new(i)    ! move em out
         enddo
        enddo
      enddo
c
c     Now finally,finally calculate Doppler shifted color
c
       bottom = (1.0+angle)+1.0e-6          ! neg sign of v receeding cube
        hue  =  hue0*(color_norm)*(gamma/bottom)
       hue = 0.75- hue
       if(hue.gt.1.0)hue = 1.0              ! protect from Dore freakout
c
c     At t=0 hue=0.25 or green. If theta=0 and v= vmax (.999999c)
c           then alog() is 12.3 and color is maxed out at 0.50
c           If v=-vmax then hue = 0.25-0.25 or blue
c
c
c           ****************************************
c
      return                    ! to graphics program
      end

      subroutine  create(surface,Lorentz_surf,beta,gamma)
c
c     Given beta and assume center of cube is at (0,0,0) find all the
c     surface points ( six surfaces 16 points on each ) surface
c     Surfaces1-4 are Lorentz contracted, beta is normal to surface 6.
c     beta is along the x direction. y direction is vertical
c
      real   surface(6,16,3)          ! pts on surface of cube
```

```
      real   Lorentz_surf(6,16,3)    ! array of Lorentz contracted pts
      real  x,y,z,xc                  ! points on surfaces
      real   lil_side,side,contside,c,gamma,alpha
c                                ! start with cube at origin
c     and assume that v points along z axis.. Then rotate cube so
c     that its contracted face is parallel to v
c
      lil_side = 12.0                 ! length of side of cube
      side = 3.0*lil_side             ! for even spacing along each side
      contside = side*gamma           ! Lorentz contracted side
c
c     Create all points on the surface if beta is along x
c
      do is = 1,4                     ! surfaces parallel to beta
       ipnt = 0                       ! init counter
       do ix = 1,4                    ! 4 perp surfaces
        y = float(ix-1)*lil_side -0.5*side
        do iy = 1,4
         ipnt = ipnt + 1              ! step to 16 for each is
          if(mod(is,2).eq.0)then      ! face 2 and 4 or bot and top
c                                     ! at +/- y = side/2
          z  = float(iy-1)*lil_side - 0.5*side
          xc = (float(is)-3.0)*0.50*gamma*side    ! contracted bot
c                                                 ! surface is # 2
          x = (float(is)-3.0)*0.50*side       ! bot surface is=2
         elseif(mod(is,2).eq.1)then           ! face 1,3  is=1,3
          z = (float(is/2) -0.5)*side         ! is =1 get -L/2,is=4 get L/2
          xc  = float(iy-1)*gamma*lil_side - 0.5*contside
          x  = float(iy-1)*lil_side - 0.5*side
         endif
         surface(is,ipnt,1) = x
         surface(is,ipnt,2) = y
         surface(is,ipnt,3) = z
         Lorentz_surf(is,ipnt,1) = xc
        do i = 2,3                        ! contraction is along x only!!
          Lorentz_surf(is,ipnt,i) = surface(is,ipnt,i)
         enddo
        enddo                             ! iy
       enddo                              ! iz
      enddo                               ! is = 1,4
c
```

```
      do is = 5,6                ! furthest (5) and closest to beta (6)
       y = side*(float(is)-5.0) - 0.50*side
      ipnt = 0
       do iz = 1,4
        z  = float(iz-1)*lil_side - 0.5*side
        do iy = 1,4
         ipnt = ipnt + 1
          x  = float(iy-1)*lil_side - 0.5*side
          xc  = float(iy-1)*lil_side*gamma - 0.5*contside
          surface(is,ipnt,1) = x
          surface(is,ipnt,2) = y
          surface(is,ipnt,3) = z
          Lorentz_surf(is,ipnt,1) = xc
         do i = 2,3                  ! contraction is along x only!!
           Lorentz_surf(is,ipnt,i) = surface(is,ipnt,i)
          enddo
         enddo
        enddo
      enddo                                ! is
c
      return
      end

      subroutine  axis_of_rot(r_surf,v,beta,rotme)
c
c     Given the particle velocity and vector from each point on
c     its surface to eye() find rotation axis
c     This is done using the vector cross product vXr_surf
c
      real  r_surf(6,16,4)           ! points on surface
      real  v(3)                     ! cube velocity
      real  rotme(6,16,3)            ! rotation axis for each point
c
      do is = 1,6                    ! six surfaces
       do ipnt = 1,16                ! 16 pnts on each
         rotme(is,ipnt,1)  =  v(3)*r_surf(is,ipnt,2)-v(2)*r_surf(is,ipnt,3)
         rotme(is,ipnt,2)  =  v(1)*r_surf(is,ipnt,3)-v(3)*r_surf(is,ipnt,1)
         rotme(is,ipnt,3)  =  v(2)*r_surf(is,ipnt,1)-v(1)*r_surf(is,ipnt,2)
       enddo
      enddo
      return
      end
```

```
      subroutine  nearest_pt(r_surf,nearest,eye,eye_mag)
c
c     Finds nearest point to observers's eye and keeps track of it
c
      real  r_surf(6,16,4)          ! scratch array
      real  nearest(4)              ! returned position and magnitude
      real  eye(3),eye_mag
      real  lorentz_surf(6,16,3)
      real  a,bmag,delta_r(3)       ! used in finding nearest
      real  magn,magna              ! scratch variables
      real  saveme(2,16)            ! scratch array
      integer  nearmag              ! number of identical nearest pts
c
      magna = r_surf(1,1,4)
      do is = 1,6
       do ipnt = 1,16
         if(magna.gt.r_surf(is,ipnt,4))then
         magna = r_surf(is,ipnt,4)
        endif
       enddo
      enddo
      nearmag = 0
      do is = 1,6
       do ipnt = 1,16
         if(magna.eq.r_surf(is,ipnt,4))then
c     Allow for possibility of more than one nearest point!
          nearmag = nearmag + 1
          saveme(1,nearmag) = is
          saveme(2,nearmag) = ipnt
        endif
       enddo
      enddo
c
      do i = 1,5                ! initialize
       nearest(i) = 0.0
      enddo
c     Calculate average nearest point
      do j = 1,nearmag
       do i = 1,4
         nearest(i) = r_surf(saveme(1,j),saveme(2,j),i)+nearest(i)
        enddo
      enddo
      do i = 1,4
```

```
        nearest(i) = nearest(i)/float(nearmag)
      enddo
c     Calculate actual nearest point starting with average nearest
c      point as an estimate
c     The following loop could be repeated to improve accuracy - but
c      one pass gives an estimate adequate for most purposes.
c
      do is=1,6                ! loop over surfaces
      do ipnt = 1,16           ! 16 points per surface
c
c   Find projection of each vector from 'nearest' to a point on surface
c  ( delta_r) and test if it is negative - if yes make new 'nearest'
c    which is perpendicular to delta_r
c
        a=0.
         do i=1,3
           delta_r(i) = r_surf(is,ipnt,i) - nearest(i)
          a = delta_r(i)*nearest(i) + a
         enddo
        if ((a.lt.0.0).and.(a .lt. -0.1)) then
c    Compute the square of the magnitude of delta_r
            bmag = 0.
            do i=1,3
             bmag=delta_r(i)*delta_r(i) + bmag
            enddo
c     compute new nearest
            do i=1,3
              nearest(i)=nearest(i)-a*delta_r(i)/bmag
            enddo
          endif
         enddo
      enddo
c
      magn = 0.0
      do i=1,3
       magn = nearest(i)*nearest(i) + magn
      enddo
      magn = sqrt(magn)
      nearest(4)=magn              ! magnitude of nearest vector
 100  return
```

```
c
c     Given cube at origin. Move to position(3) and then
c     rotate it to point along the beta direction
c     returns rotated array in surface
c
      real   beta,v(3)          !  v/c, velocity vector of cube
      real   surface(6,16,3)    !  original/rotated array
      real  theta
      real  little              !  blowup protection
      real  A1,B1,C1            !  vector components of curl
      real  V1V,L               !  in transform matricies
      real  x1,x2,x3            !  temporary x position
      real  y1,y2,y3            !  temporary y position
      real  z1,z2,z3            !  temporary z position
      real  sinI,cosI,sinJ,cosJ !  angles for rotation about VXrip
      real  vector(3)           !  vector position of each pt on cube
c
      little = 1.0e-6                 !  to prevent blowup
c
c     Determine axis of rotation.
c
      A1 = 0.                         !  components of rotation axis
      B1 =  v(3)/(beta + little)
      C1 = - v(2)/(beta + little)
c
        V1V = (B1*B1 + C1*C1)
        L = sqrt(V1V + A1*A1)
         V1V = sqrt(V1V)
        cosI = C1/(V1V + little)
        sinI = B1/(V1V + little)
        cosJ = V1V/(L + little)
        sinJ = A1/(L + little)
c
      if((C1.eq.0.).and.(B1.eq.0.)) goto 100
c     Rotation angle is found by taking dot product between y-axis
c     and v.
c
       theta = acos(v(1)/(beta + little))
c
      do isur = 1,6
       do ipnt = 1,16
        do iu = 1,3
          vector(iu) = surface(isur,ipnt,iu)
```

```
        enddo
c
        x1 = vector(1)
        y1 = cosI*vector(2)-sinI*vector(3)
        z1 = cosI*vector(3)+sinI*vector(2)
c
c        now rotate cube about y so that z axis corresponds to axis of
c           rotation
c
        x2 = cosJ*x1-sinJ*z1
        y2 = y1
        z2 = cosJ*z1+sinJ*x1
c
c     now DO the relativistic rotation  ( about new z)
c
        x3 = x2*cos(theta) + y2*sin(theta)
        y3 = y2*cos(theta) - x2*sin(theta)
        z3 = z2
c
c     now do inverse transforms
c
        x2 = cosJ*x3+sinJ*z3              ! inverse rot about y
        y2 = y3
        z2 = cosJ*z3-sinJ*x3
c
        x1 = x2                       ! inverse rot about x
        y1 = cosI*y2+sinI*z2
        z1 = cosI*z2-sinI*y2
c
c     Now translate cube back to where it was at the outset
c
        surface(isur,ipnt,1) = x1
        surface(isur,ipnt,2) = y1
        surface(isur,ipnt,3) = z1

c
        enddo                             ! ipnt over points
      enddo                               ! isur  6 surfaces
c
100   return
      end
c
      subroutine  calc_r(Lor_surf,r_surf,beta,eye)
```

```
c
c     Calculates the angle of rotation
c     for each point on surface according to apparant relativistic
c     rotation. Taylor Introductory Mechanics pg 357
c     These are put into an array (sigh(6,16)) to be used later
c
      real  beta
      real  eye(3)              ! position of observer's eye
      real  Lor_surf(6,16,3)    ! points on surface of rotated cube
      real  little              ! overflow prevention
      real  r_surf(6,16,4)      ! for calculations (rpoint,rmag)
      real  rpoint(3),mag_rpoint
c
      if(beta.eq.0)goto  100          ! bail out no need to work
      little = 1.0e-6                 ! divide protect
c
      do is = 1,6               ! over all 6 surfaces
       do ipnt = 1,16                 ! 16 points/surface
c
c     Find the spatial location of points on the surface
c     as measured from the observer's position.
c
         mag_rpoint = 0.0       ! initialize
        do i = 1,3
          rpoint(i) = Lor_surf(is,ipnt,i) - eye(i)
          mag_rpoint = mag_rpoint + rpoint(i)**2
         enddo
          mag_rpoint = sqrt(mag_rpoint)
c
c     Calculate angle of elevation for this x,y,z triplet
c
        do i = 1,3
          r_surf(is,ipnt,i) = rpoint(i)             ! save for later
         enddo
          r_surf(is,ipnt,4) = mag_rpoint
        enddo                                       ! ipnt
       enddo                                        ! is
100   return
      end
```

```
c
      real  r_surf(6,16,4)            ! store scratch rpoint,magr,sigh
      real  ph_surf(6,16,3)           ! light surface - position of
c                                     ! retarded emission points
      real  r_ret_hat(6,16,3)         ! unit vector along r-retarded
      real    beta,beta2,v(3),eye(3)
      real  rmag                      ! magnitude of  rpoint()
      real  nearest(4)                ! nearest point on cube to eye
      real  sigh_tot(6,16)            ! total rotation angle
      real  c,d,little
      real  a1,b1,c1,d1               ! for calculating rret
      real   r_dot_beta,b2rn,rnmag,rret
      real  vt,ret_pos(3),cos_sigh
c
c     calculate angle beta makes with x-z plane
c
      little = 1.0e-6                 ! divide protect
      if(beta.eq.0)goto  100          ! bail out no need to work
      beta2 = beta*beta
      rnmag = nearest(4)
      b2rn = beta2*rnmag
c
      do is = 1,6                ! over all 6 surfaces
       do ipnt = 1,16                 !  16  points/surface
      rmag = r_surf(is,ipnt,4)
c
c  Calculate  the position of the points such that
c   emitted rays reach the observer simultaneous
c  with   a ray from the nearest point.
c
        r_dot_beta = 0.0
       do i = 1,3
         r_dot_beta = r_surf(is,ipnt,i)*v(i) + r_dot_beta
       enddo
        a1 = 1. - beta2
        b1 = 2*(r_dot_beta + b2rn)
         c1 = -rmag*rmag  -  2.*r_dot_beta*rnmag-b2rn*rnmag
        d1 = b1*b1 - 4.*a1*c1
        rret = (-b1 + sqrt(d1))/(2.*a1 + little)
        vt = rret-rnmag
       cos_sigh = 0.0
      do i = 1,3
       ret_pos(i) = r_surf(is,ipnt,i)  -vt*v(i)
```

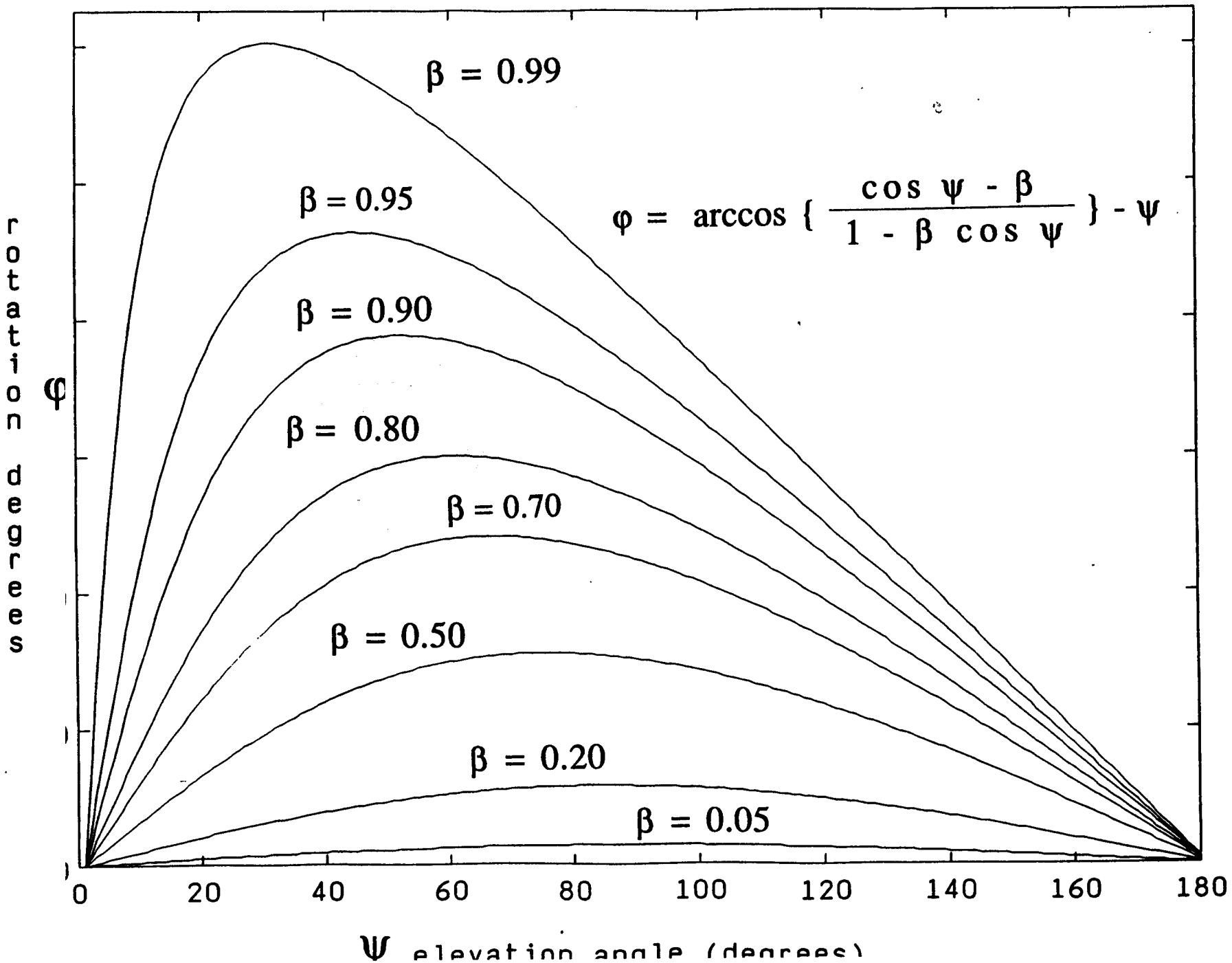
```
        r_ret_hat(is,ipnt,i) = ret_pos(i)/(rret + little)
        ph_surf(is,ipnt,i) = eye(i) + ret_pos(i)
c
c     Calculate the angle of elevation of the retarted position.
c
         cos_sigh = - ret_pos(i)*v(i) + cos_sigh
        enddo
         cos_sigh = cos_sigh/(rret*beta + little)
        sigh_tot(is,ipnt) = acos(cos_sigh)
c
       enddo                          ! ipnt
      enddo                           ! is
c
100   return
      end

      subroutine  relative(surface,nearest,sigh_tot,r_surf,rotme,
      1           r_ret_hat,eye,v,beta,pos_new,angle)
c
c     Rotate each point on surface according to apparant relativistic
c     rotation. Taylor + plane wave correction
c     With sigh_tot the total rotation angle move cube so that axis
c     through nearest point is z axis and rotate each point by sigh_tot
c     about this. rotate pointson the uncontracted cube. Points on the
c     contracted cube were used to find the rotation angles
c
      real  r_surf(6,16,4)           ! store scratch rpoint,magr,sigh
      real  beta,v(3),beta2,gamma    ! v/c,velocity, beta*beta
      real   surface(6,16,3)         ! points on surface of rotated cube
      real  nearest(4)               ! nearest point on cube to eye
      real  sigh_tot(6,16)           ! total rotation angle
      real  rotme(6,16,3)            ! rotation axis
      real  r_ret_hat(6,16,3)        ! unit vector along r-retarded
      real  little                   ! small number
      real  eye(3),pos_new(3)        ! dist eye to (o,o,o) and new pos
      real  ctr_to_nearest(3)        ! vector for moving cube b/4 rot
      real  ctn_unc(3)               ! uncontracted ctr_to_nearest
      real  proj(3),pro              ! projection along beta
      real A1,B1,C1                  ! vector components of curl
      real V1V,L                     ! in transform matricies
      real  theta,xx,angle           ! cos(total angle of elevation)
```

```
      real  sign,signdop              ! sign's of angles
      real  sinI,cosI,sinJ,cosJ       ! angles for rotation about VXrip
      real  vector(3)                 ! vector pos of each pt on cube
      real  r_dot_rhat                ! dot product of r and r_ret_hat
c
      if(beta.eq.0.0)goto  100        ! no action dont bother
      little = 1.0e-6
       beta2 = beta*beta
       gamma = 1./(sqrt(1.-beta2))
       ctn_dotv = 0.                  ! used to undo a Lorentz contract
      do i = 1,3
        ctr_to_nearest(i) = nearest(i) - pos_new(i)  + eye(i)
        ctn_dotv = ctr_to_nearest(i)*v(i) + ctn_dotv
      enddo
       ctn_dotv = ctn_dotv*(gamma - 1.0)/beta2
c
c     Uncontract center-to-nearest point on cube vector
c
      do i = 1,3
        ctn_unc(i) = ctr_to_nearest(i) + ctn_dotv*v(i)
      enddo
c     Rotate each vector on the surface around rotme by angle sigh_tot
      do is = 1,6
       do ipnt = 1,16
         A1 = rotme(is,ipnt,1)
         B1 = rotme(is,ipnt,2)
         C1 = rotme(is,ipnt,3)
        V1V = (B1*B1 + C1*C1)
        L = sqrt(V1V + A1*A1)
        V1V = sqrt(V1V)
        cosI = C1/(V1V + little)
        sinI = B1/(V1V + little)
        cosJ = V1V/(L + little)
        sinJ = A1/(L + little)
c
c     Calculate angle of elevation for this x,y,z triplet
c     For Doppler shift evaluation
c
         cos_doppler = 0.0
        do i = 1,3
          cos_doppler = cos_doppler - r_surf(is,ipnt,i)*v(i)
        enddo
          cos_doppler = cos_doppler/(r_surf(is,ipnt,4) + little)
```

```
        angle = angle + cos_doppler                    ! ave angle for
c                                                  ! Doppler shift
c
c
c     Now do relativistic rotation
c
       sigh = sigh_tot(is,ipnt)
      cos_sigh = cos(sigh)
      xx = (cos_sigh - beta)/(1. - beta*cos_sigh + little)
      if(abs(xx).gt.1.) xx = 1.
      theta = acos(xx) - sigh
c
c     Before rotation about rotme we must
c     Translate cube back so that it will rotate about nearest()
c     Recalculate Lorentz contraction
c     Uncontract component of ctr_to_nearest along velocity vector
       do i = 1,3
c     Vector points from a point on the surface nearest the
c     observer to each labled point on the surface
          vector(i) = surface(is,ipnt,i) - ctn_unc(i)
        enddo
c
c        now rotate cube (about x) so that new axis of rotation
c        is in the x-z plane
c
        x1 = vector(1)
         y1 = cosI*vector(2)-sinI*vector(3)
        z1 = cosI*vector(3)+sinI*vector(2)
c
c rotate cube about y so that z axis corresponds to axis of rotation
c
        x2 = cosJ*x1-sinJ*z1
        y2 = y1
        z2 = cosJ*z1+sinJ*x1
c
c     now DO the relativistic rotation  ( about new z)
c
         x3 = x2*cos(theta) + y2*sin(theta)
         y3 = y2*cos(theta) - x2*sin(theta)
        z3 = z2
c
c     now do inverse transforms
```

```
      y2 = y3
      z2 = cosJ*z3-sinJ*x3
c
      x1 = x2                          ! inverse rot about x
      y1 = cosI*y2+sinI*z2
      z1 = cosI*z2-sinI*y2
c
c     Correct rotated surface so that each point lies along direction
c     of corresponding point on photosurface.
c
      r_dot_rhat = (nearest(1) + x1)*r_ret_hat(is,ipnt,1)
      r_dot_rhat = (nearest(2) + y1)*r_ret_hat(is,ipnt,2)
      1                         + r_dot_rhat
      r_dot_rhat = (nearest(3) + z1)*r_ret_hat(is,ipnt,3)
      1                         + r_dot_rhat
      x1 = r_dot_rhat*r_ret_hat(is,ipnt,1) - nearest(1)
      y1 = r_dot_rhat*r_ret_hat(is,ipnt,2) - nearest(2)
      z1 = r_dot_rhat*r_ret_hat(is,ipnt,3) - nearest(3)
c
c     Now translate cube back to where it was at the outset
       surface(is,ipnt,1) = x1   + ctr_to_nearest(1)
       surface(is,ipnt,2) = y1   + ctr_to_nearest(2)
       surface(is,ipnt,3) = z1   + ctr_to_nearest(3)
c
       enddo                           ! ipnt
      enddo                            ! is
100   angle = angle/(16.0*6.0)  ! ave angle cube makes with x-z plane
      return
      end
```

Figure 2


β = 0.99
β = 0.95
β = 0.90
β = 0.80
β = 0.70
β = 0.50
β = 0.20
β = 0.05
φ = arccos { (cos ψ - β) / (1 - β cos ψ) } - ψ
φ
rotation degrees
0
20
40
60
80
100
120
140
160
180
ψ elevation angle (degrees)

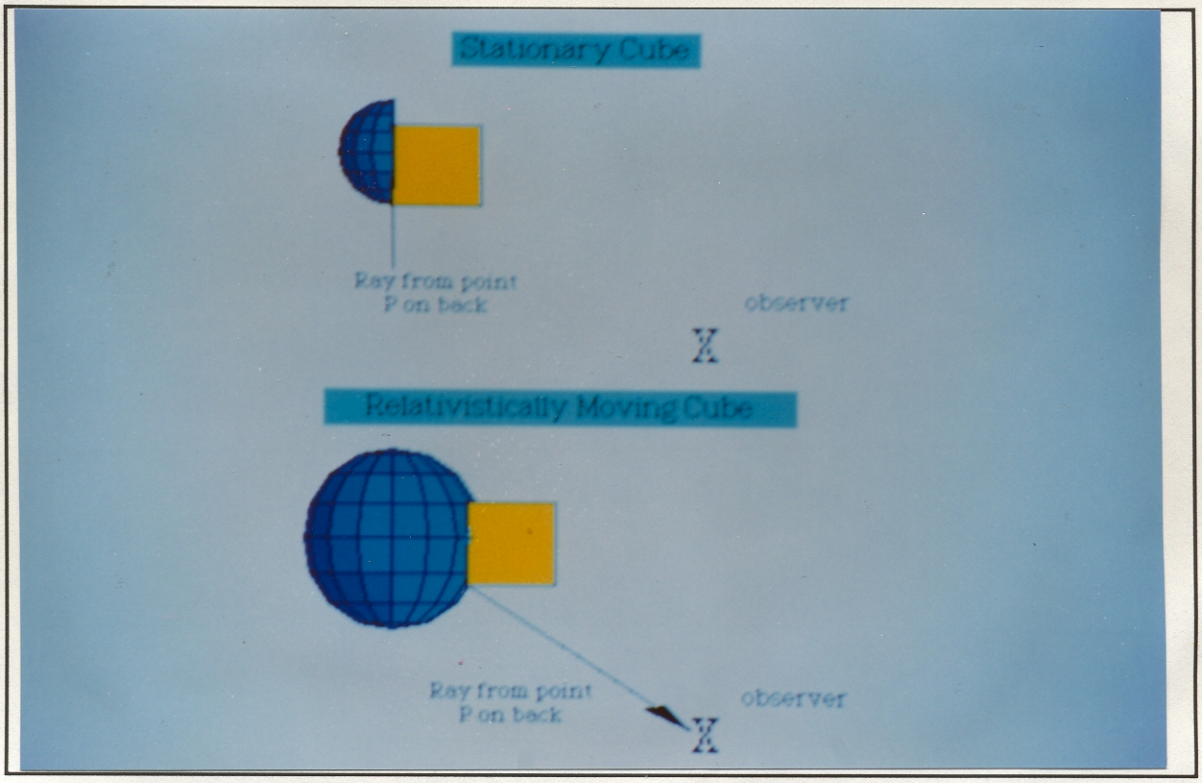


Figure 1

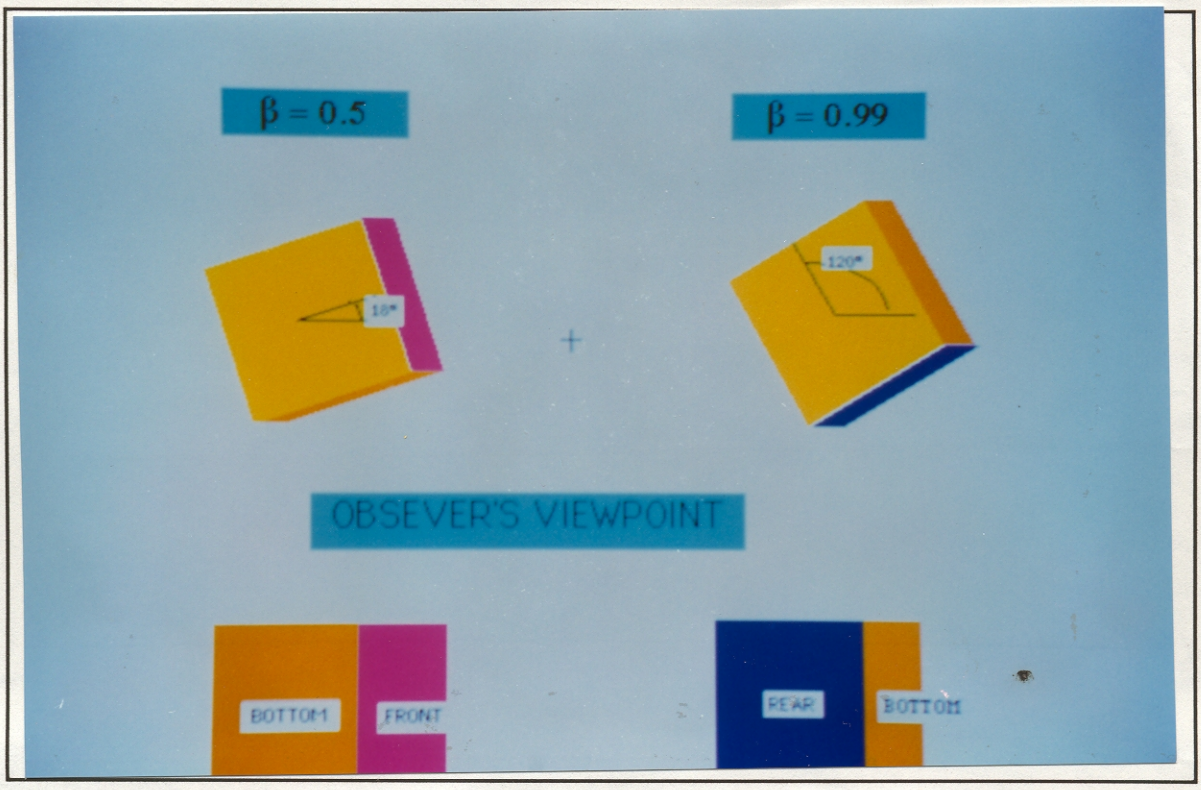


Figure 3

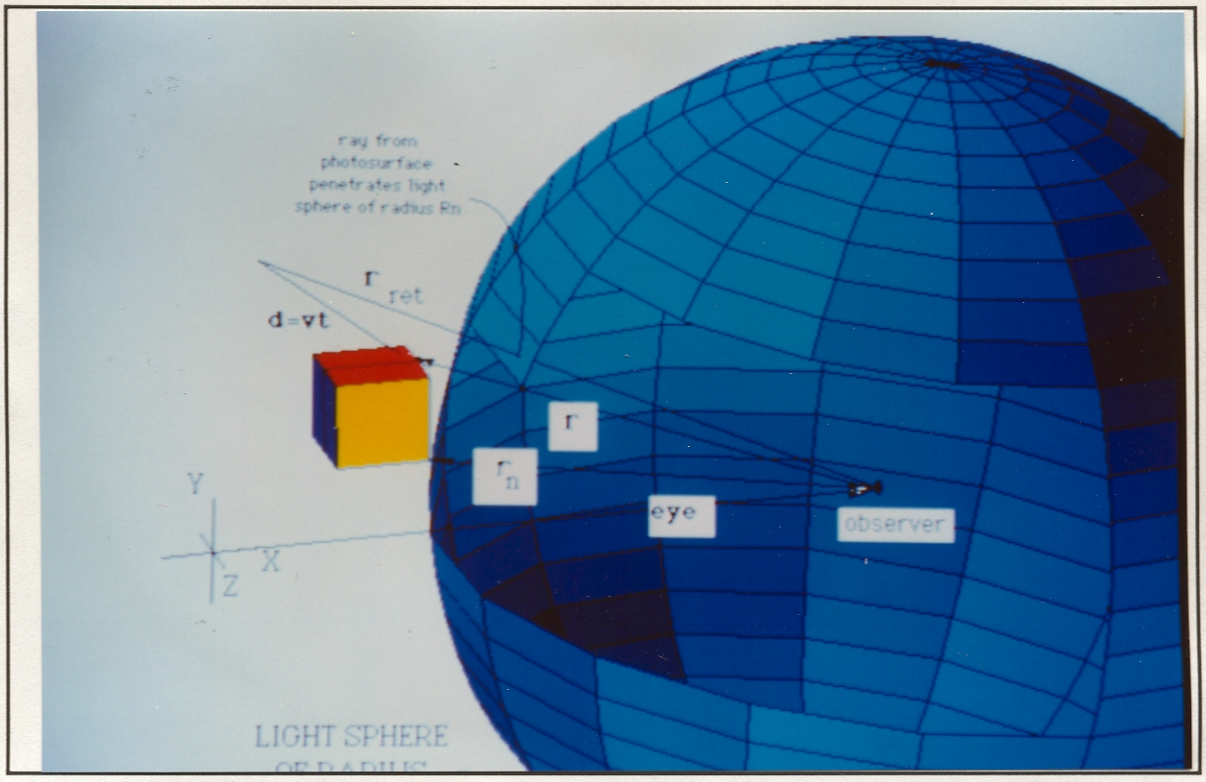


Figure 4

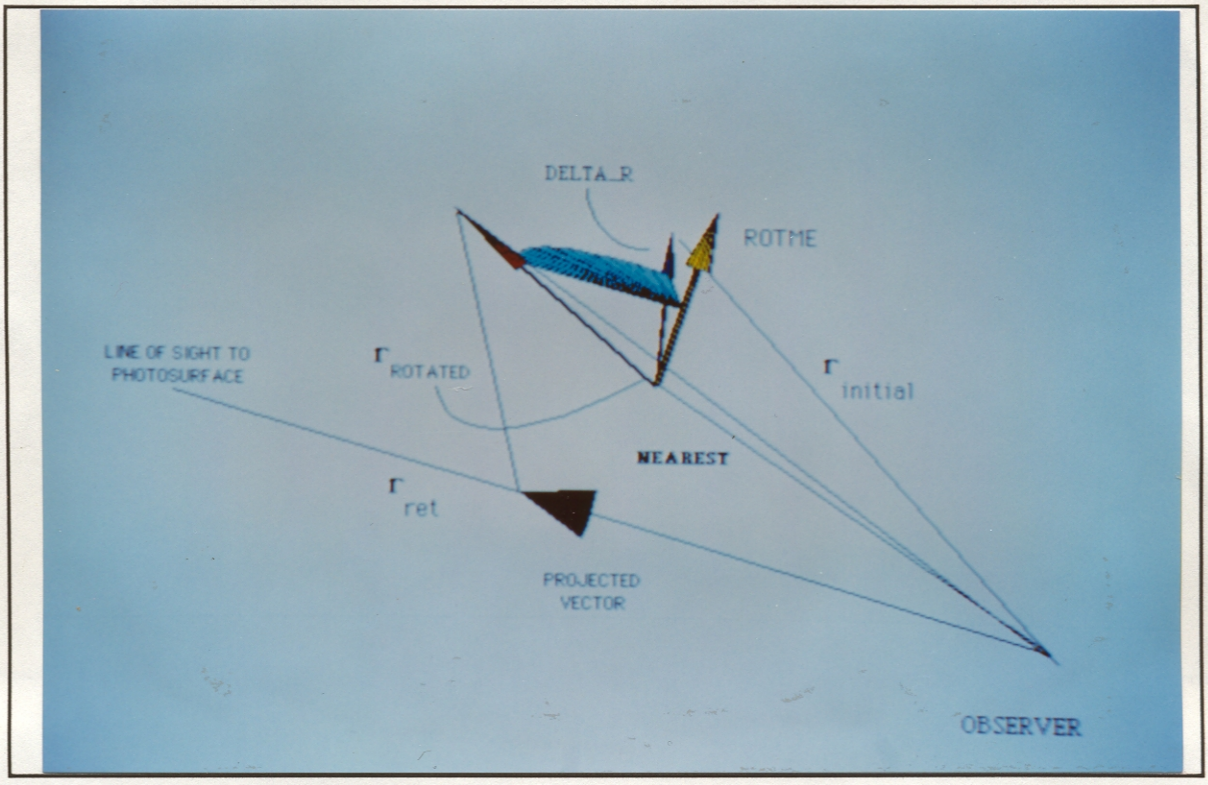


Figure 5

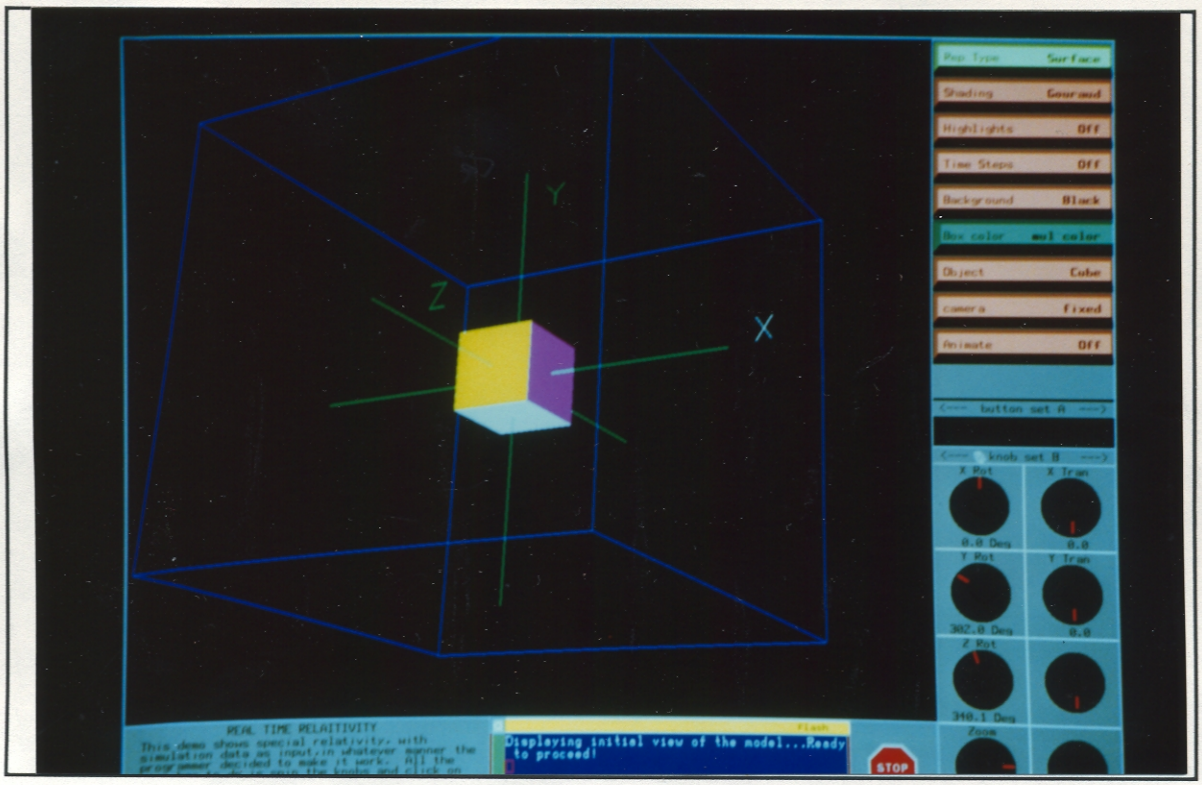


Figure 6

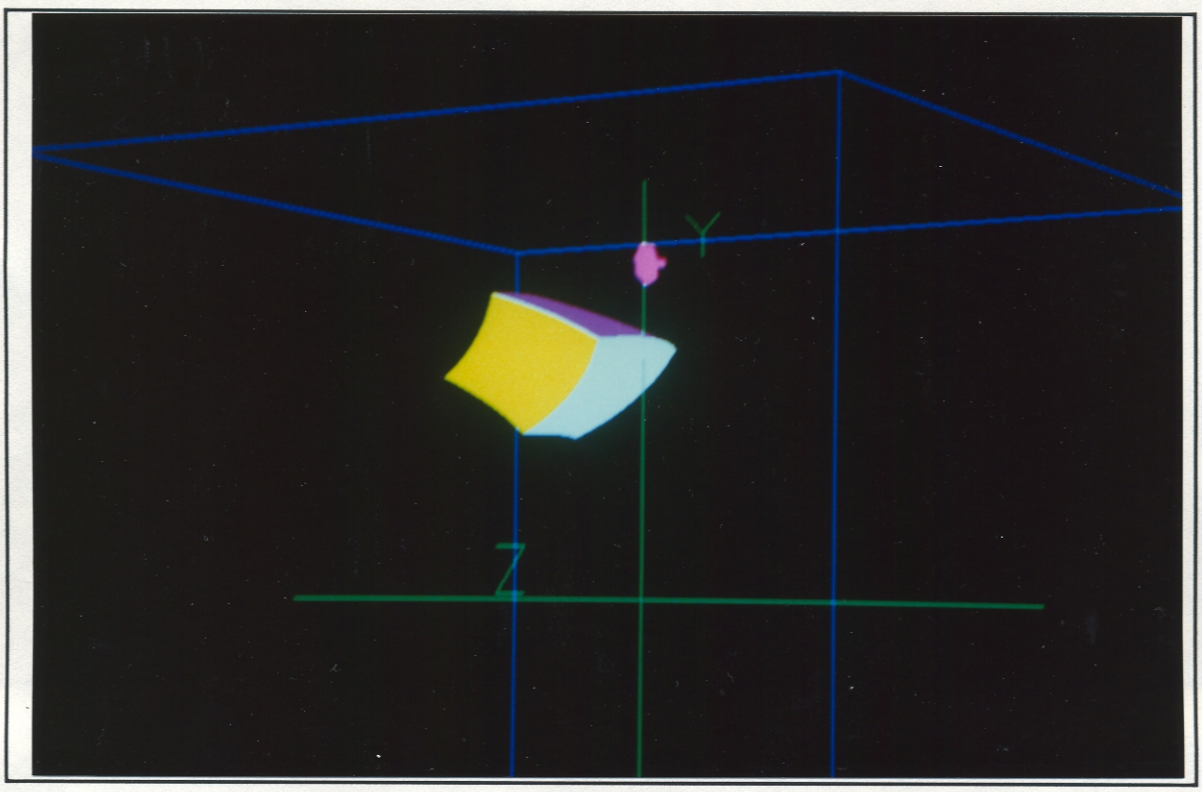


Figure 7a

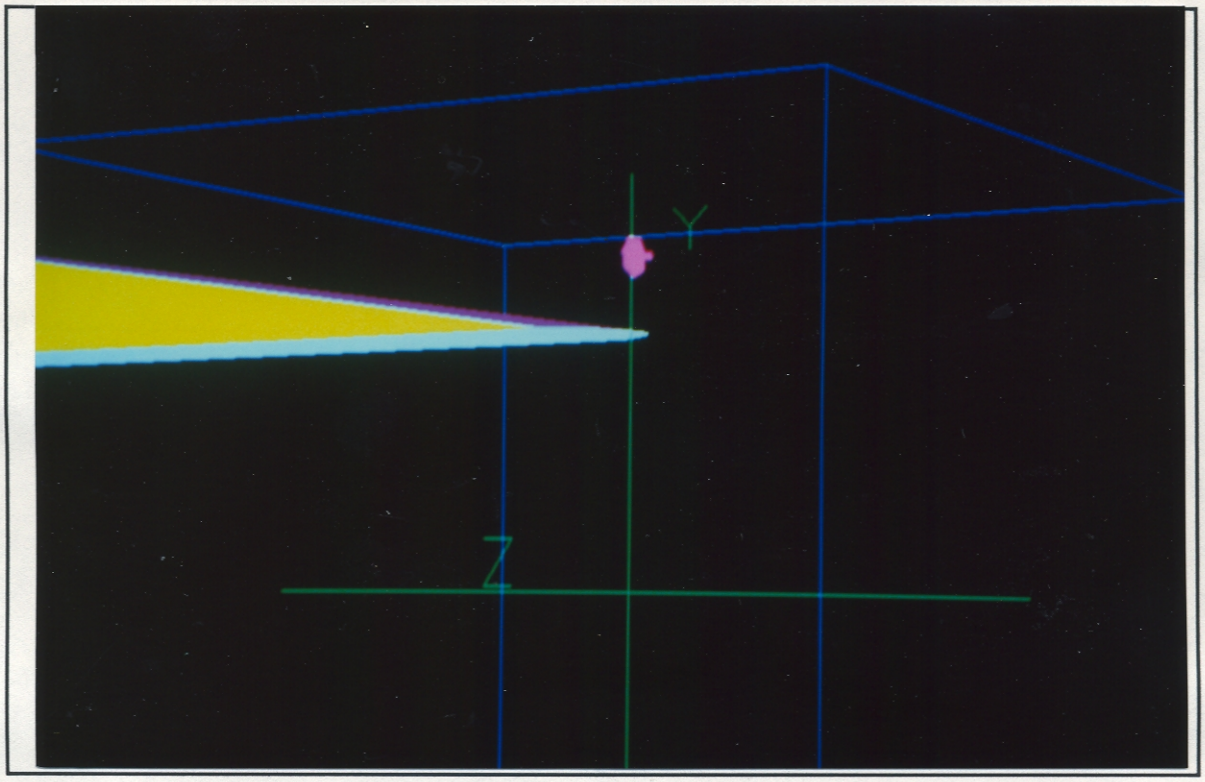


Figure 7b

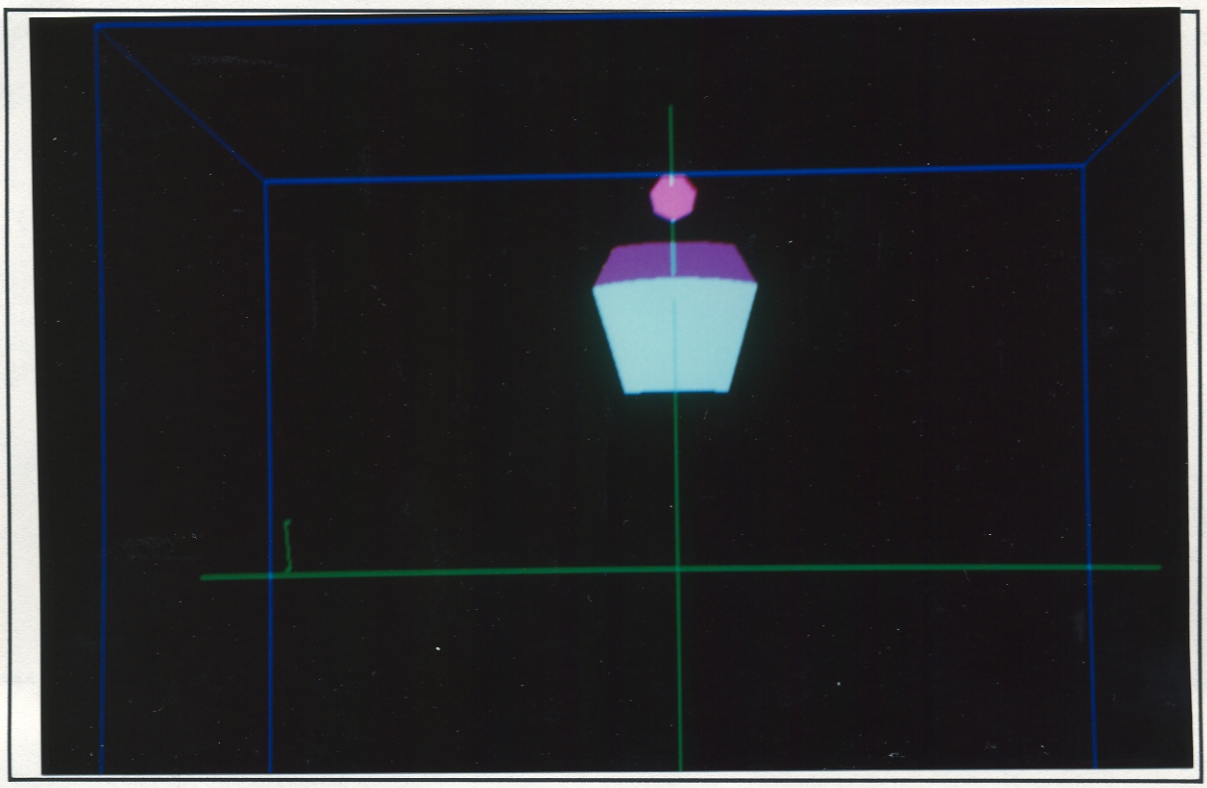


Figure 7c

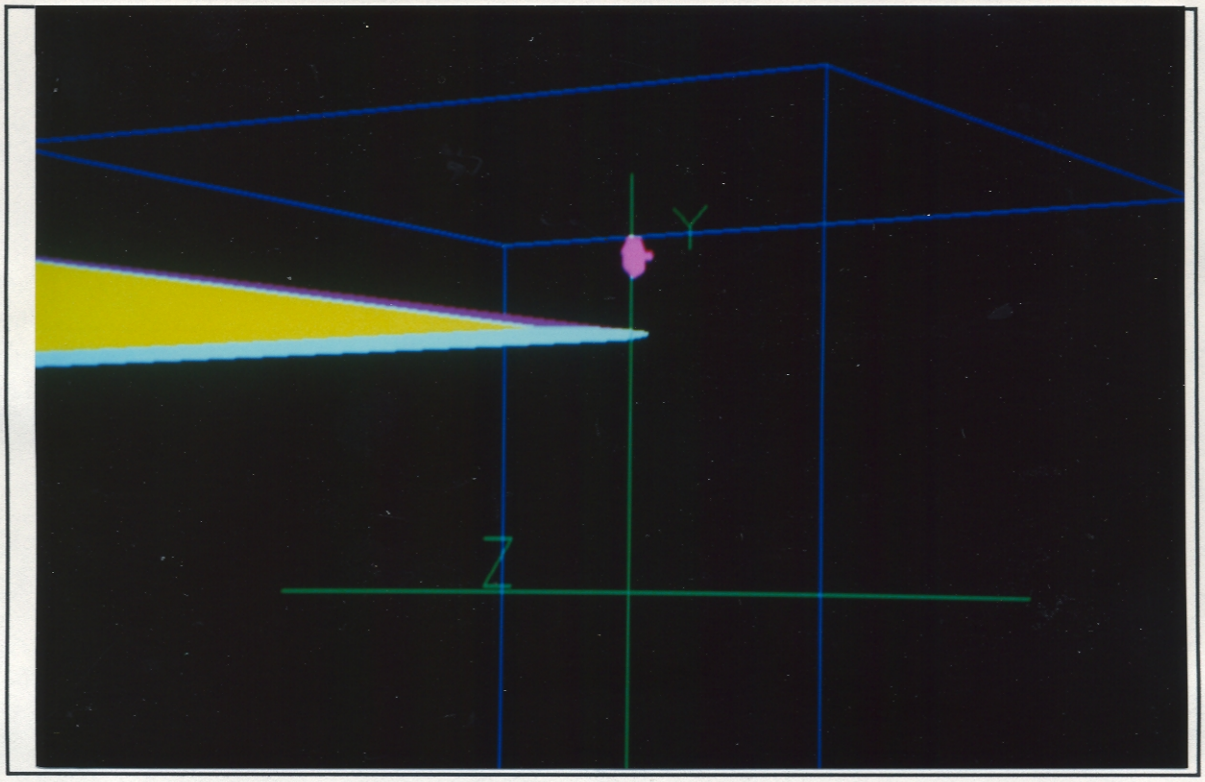


Figure 7b

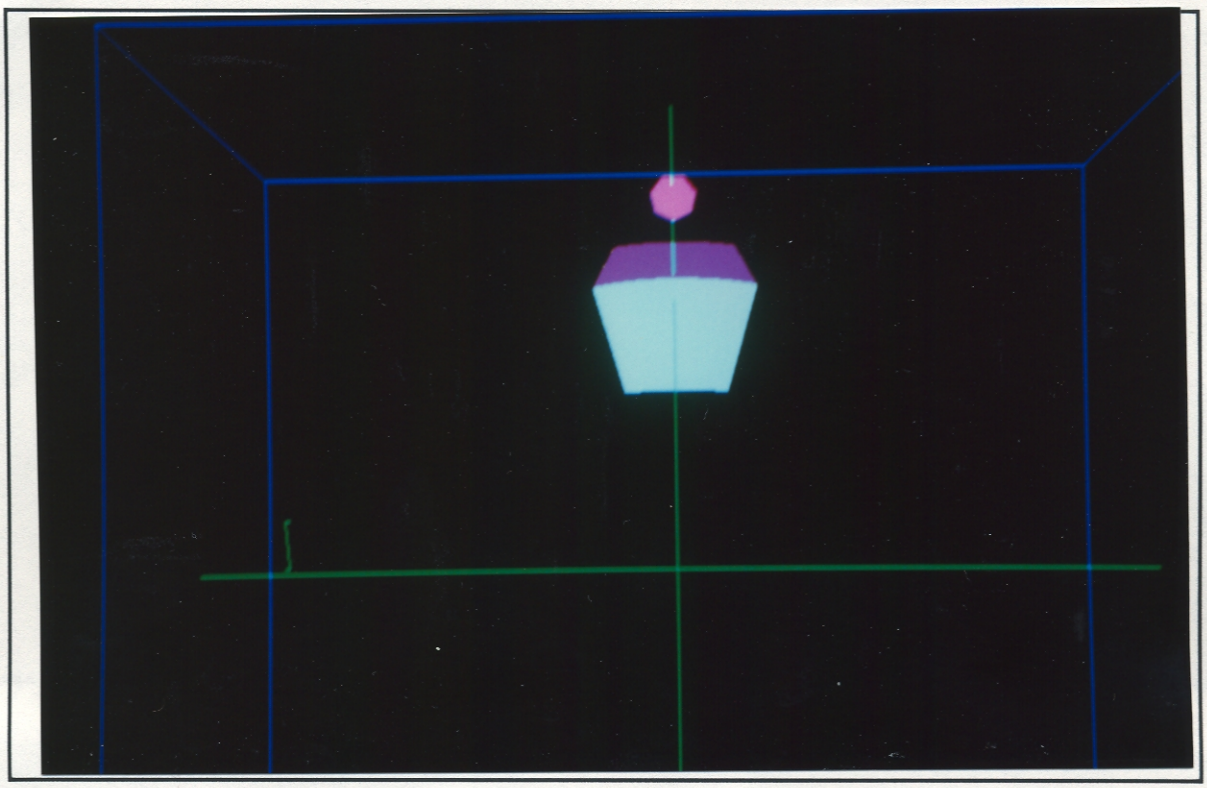


Figure 7c

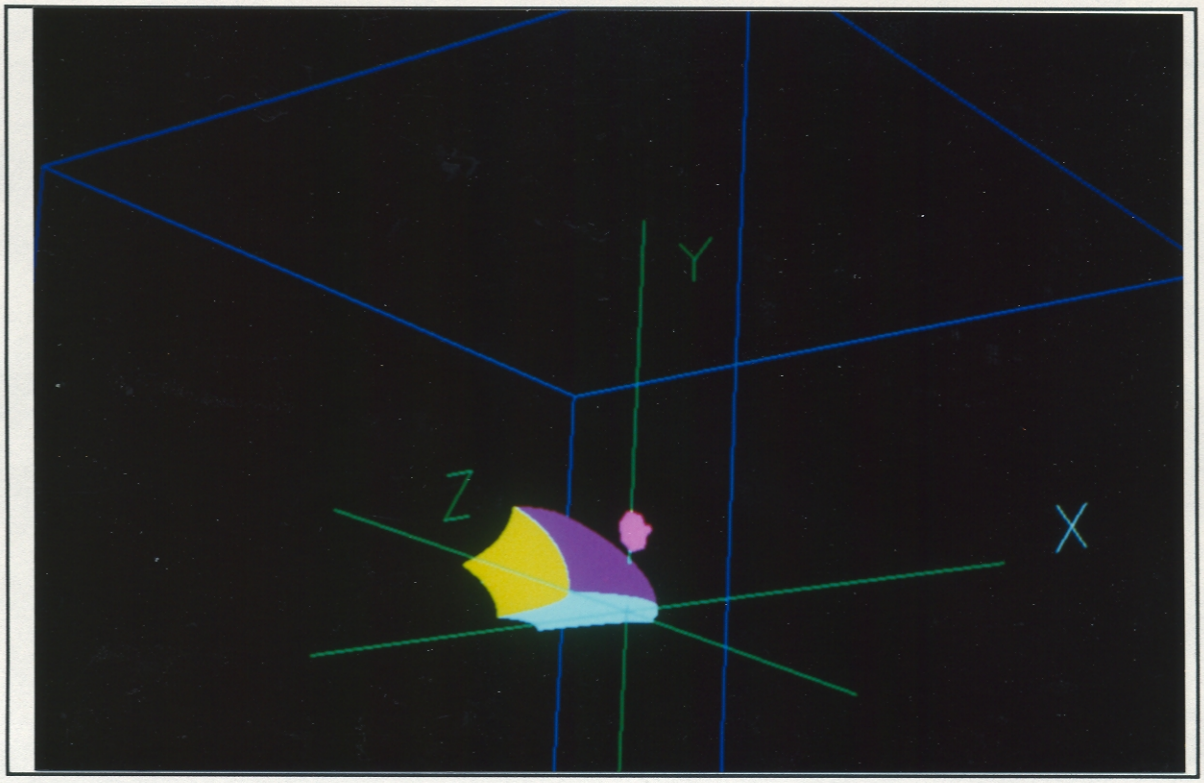


Figure 8

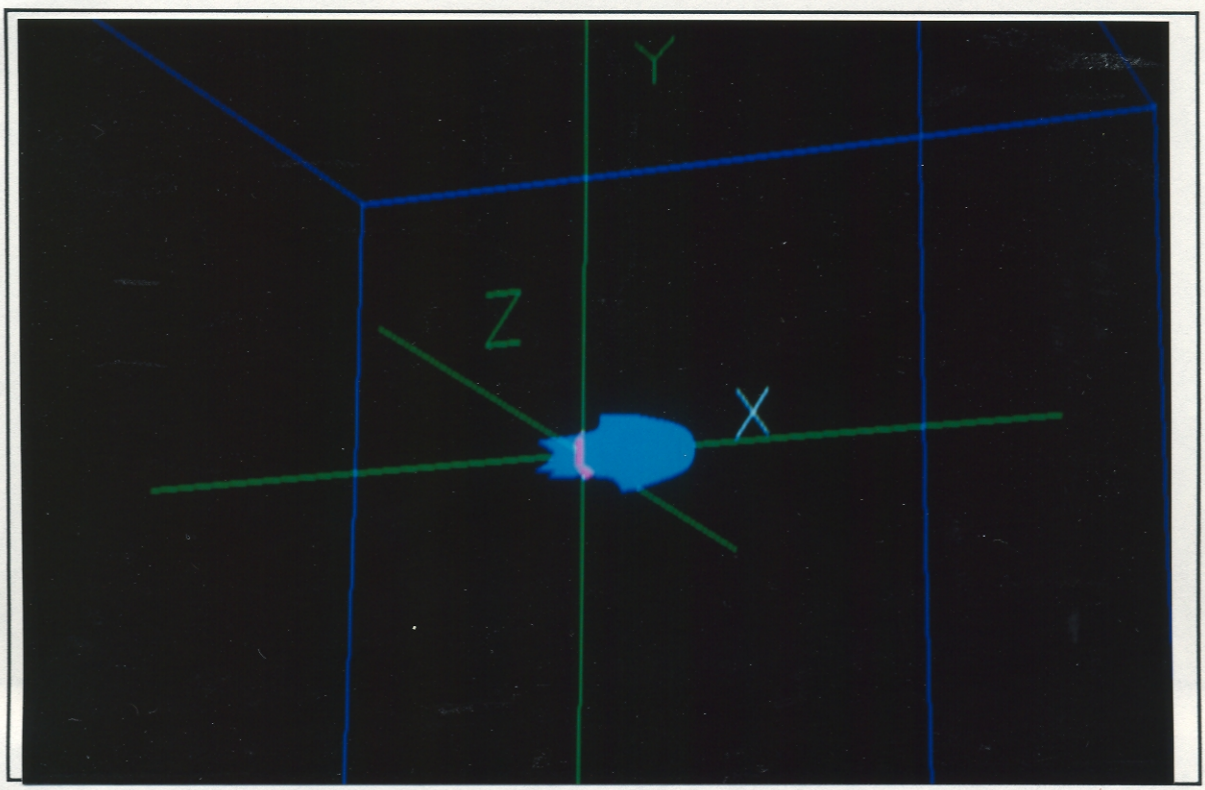


Figure 9a

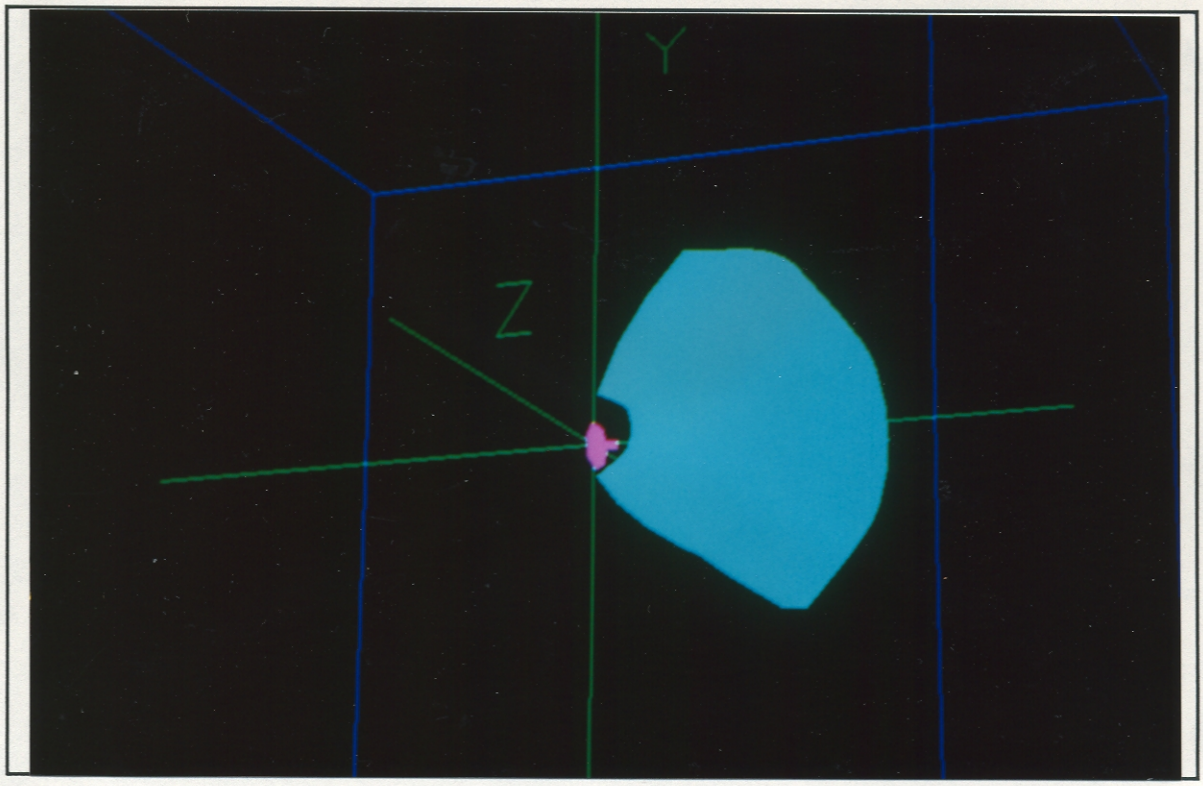


Figure 9b

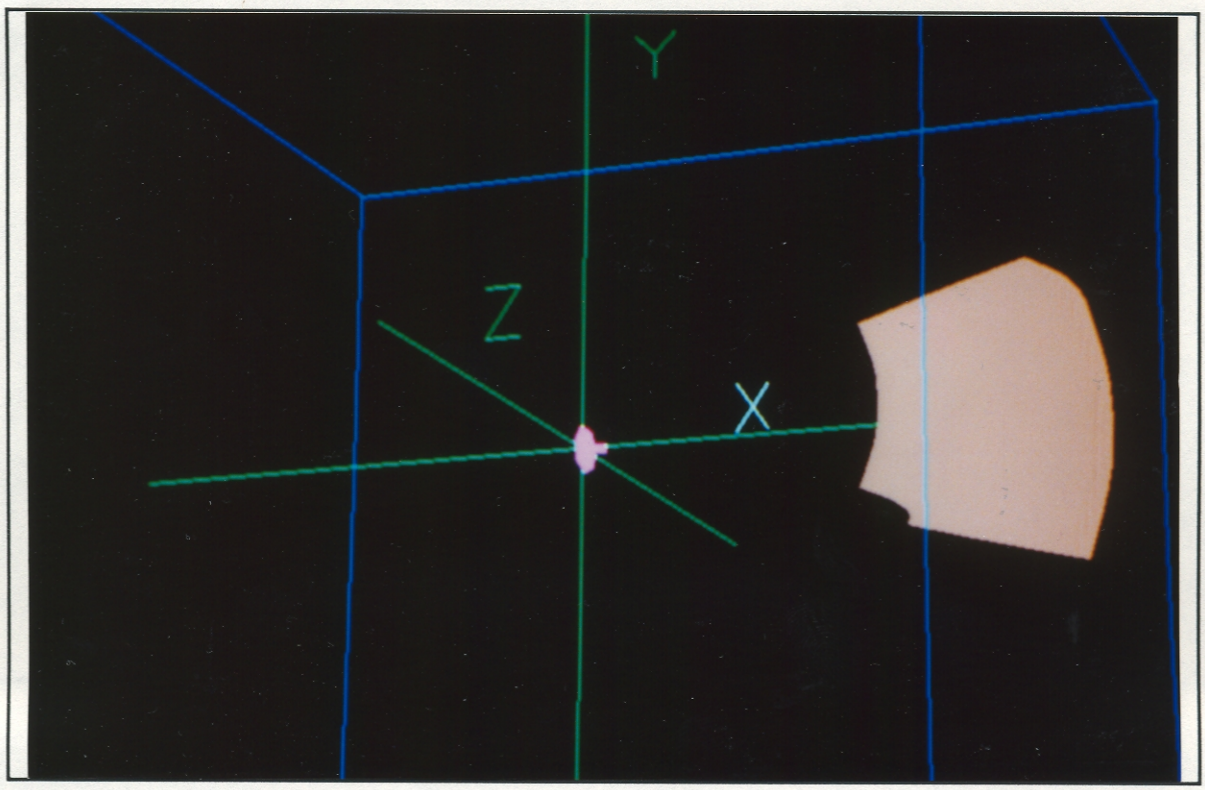


Figure 9c

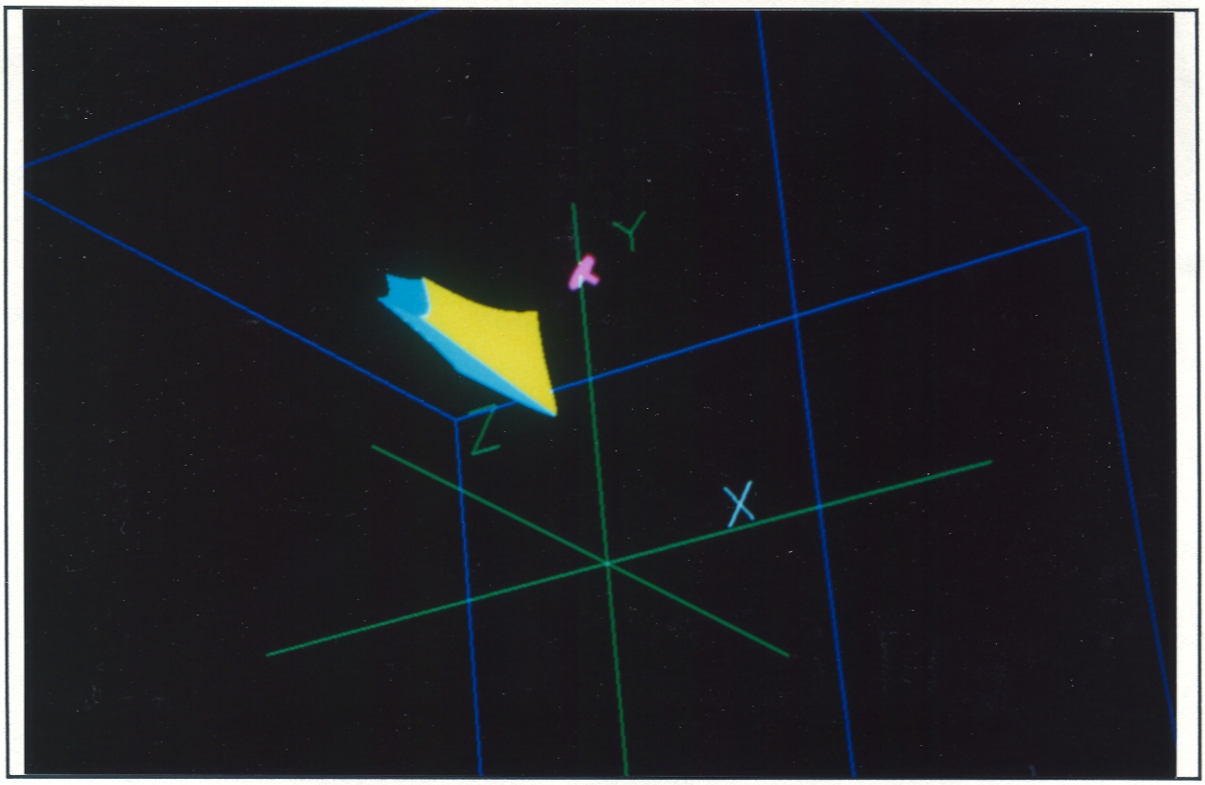


Figure 10