RPI Image Processing Laboratory COMPUTER DOCUMENTATION	Bulletin No. D-137	Rev. 1.0	Page 1

TITLE: The Straw User's Manual

ABSTRACT: This bulletin describes how to operate the IPL's raytracer known as Straw. While Straw will produce realistic images it is very costly to run. It is useful to those who would like to produce graphics for computer vision or for graphics own sake.

DESCRIPTION:

To run Straw the user types:

```
SEG STRAW>STRAW
```

It is usual for the user to run Straw as a phantom. To get started running your Straw programs a special section entitled PRIMOS exists for the first time user, this begins on page 69.

[Use additional sheets if necessary]

Prepared by: Douglas Lyon	Date: 8/17/87	Total Pages: 77

Abstract

This report describes how to use a software system for generating pictures on the Deanza. The software is known as Straw. The technique for generating the pictures is known as raytracing. Raytracing enables a user to set up a scene which looks real. The process of defining the scene to the computer is not well defined. Because of the somewhat limited interface a keyboard has to offer the user is often involved in stating geometries and cooefficients to the Straw program. The Straw programming language is a syntax which allows the user to concentrate effort on the geometries and cooefficients without having to deal to much with the computer. This report will hopefully bring raytracing to the non-graphics-specialist.

Table of Contents

Acknowledgment

The research reported here was supported by the Image Processing Laboratory, Professor Herbert Freeman, Principal Investigator.

Introduction

This is a users manual for the in house raytracer used at the Image Processing Laboratory at RPI. A raytracer traces a line from a source of light to a raster element in a display (called a pixel) and determines the pixels brightness by various techniques. The user of this software need not be concerned with how the program works. This manual will not cover the theory behind raytracing. Whenever an intuitive feel for a coefficient is needed an example photograph is presented which should help the user understand the meaning of a term. A classic example is the quadric surface. Experimentation is required for the user to get a feel for the shapes derivable for the surface. In the appendix on Examples of the use of the quadric_coeffs Command the user will find all families of quadric surfaces presented so that a quadric surface grammar may result. It is expected that the user will become familiar enough with the effects of the command of interest so that only the reference part of this manual needs to be used.

Getting Started

In order to get started in straw you must first read the tutorial on your operating system. Each operating system for which straw runs is described after the bibliography at the end of the table of contents. This manual is not ment to be read as a novel but is instead a guide to a large software subsystem which requires your interaction.

GOOD LUCK!

Notation

The Straw primitives which are intended to be used in a input command sequence and the user callable extensions to this package are described in this appendix.

Straw commands may be entered in any order within reason. Exit will have an immediate effect, you should try to mirror your own thought processes in the order that you use your commands. A geometry command (sphere, plane..) will make reference to a color which must be defined already. Never refer to a nonexistent entity. If you follow this rule you will probably not get into trouble.

Throughout this manual a consistent notation is used:

'*' - denotes multiplication.

'**' - means raised to the power, x**2 means x*x.

'dx' - denotes a change in x.

'dy' - denotes a change in y.

'dz' - denotes a change in z.

Unless otherwise noted all positional units are in millimeters, all rotational units are degrees. The units for ambience are unknown.

A common layout is utilized to describe the various primitives. The layout consists of a series of entries and the relative information. Please note that in the case of interactive dialogues the computers response is underlined. If the computer response is unknown then the symbol '..' is used to indicate that situation dependent output will be

emmitted by the computer. Reading through the layout the user will see a similarity between the documentatin style and the UNIX style, this is delibrate. A unified documenation standard is essential for easy reader interface. User friendly is not the point, rather information content and accessability have been maximised.

ROUTINE NAME The name of the primitive and a one-line description of the primitive are given.

CALLING SEQUENCE The actual calling sequence of the primitive in the user's software is displayed. Sometimes '{...}' are used to indicate a portion of the calling line which may be repeated forever. A '|' indicates that the user is given a choice of one argument OR another.

ARGUMENTS The varible utilized in the calling sequence are identified and described. Each variable is in the form:

<type> <varible> - <description>
where <type> is the type of the varible. The possible types are integer, character (data type representing character strings such as integer), and keyword (special sequence of characters which give a token to the command parser). If a varible is the return of a function, <description> will start with "Function return,".

DESCRIPTION A description of the function of the primitive is given.

IMPLEMENTATION A brief description of the algorithm utilized for the primitive is given.

DEFICIENCIES AND PROBLEMS The primivie may have some defiencies of problems that may be due to the algorithm utilized or to the dependence of the algorithm to the underlying system. This entry makes the user aware of the possible deficiencies and problems of the primitive.

ROUTINES CALLED This entry lists the routines or system-dependent macros that are utilized in the implementation of the primitive.

REFERENCES Pointers to further information on the primitive or on the algorithm utilized is given in this entry.

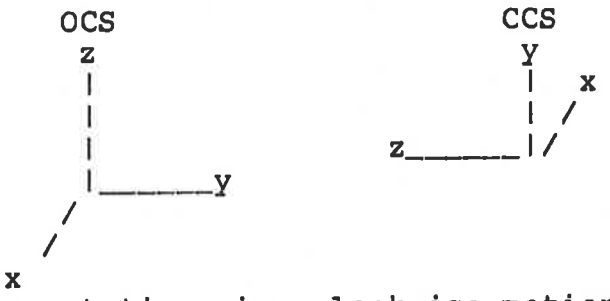
The Straw primitives are described in alphabetical order.

ROUTINE NAME CAMERA - places a camera in the object coodinate system

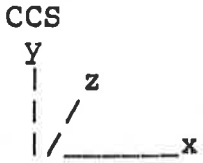
CALLING SEQUENCE Camera: <x> <y> <z> <rot_x> <rot_y> <rot_z> <focal_length>;

ARGUMENTS real x - the x position in the scene space.
real y - the y position in the scene space.
real z - the z position in the scene space.
real rot_x - the degrees of rotation about the camera's x axis.
real rot_y - the degrees of rotation about the camera's y ayis.
real rot_z - the degrees of rotation about the camera's z axis.
real focal_length - the actual focal length of the camera in millimeters.

DESCRIPTION The camera command will position the virtual camera in a scene. The camera has no physical attributes, will therefore cast no shadow and cannot be seen even in a mirror. The rotation axis are shown below, OCS is for Object Coordinate System, CCS is for Camera Coordinate System.



Positive rotation is clockwise motion about the named CCS axis as if you were looking toward the origin while sitting on the axis. Initially the camera positioned at point (0,200,0) in OCS will point towards (0,0,0) in the OCS iff dx,dy, and dz are 0. The camera therefore looks out its own z axis. This is translation and rotation independent, the camera always looks down the CCS Z axis. From the initial camera position the grammer for camera is as follows:
a change in dy is a pan.
a change in dx is a tilt.
a change in dz is a swing.
if dy = 90 degrees the CCS looks like this:



IMPLEMENTATION In order to properly rotate the camera use the following formula:
x_rot = 90 - arctan(dz / sqrt(dx**2 + dy**2))
y_rot = arctan(dx/dy)

z_rot = image rotation on the screen (+ is clockwise).
dx = x - X_Coordinate_to_be_center_of_screen
dy = y - Y_Coordinate_to_be_center_of_screen
dz = z - Z_Coordinate_to_be_center_of_screen

The focal_length is calculated by the following formula,
focal_length = frame_size / window_size * sqrt(dx**2 + dy**2 + dz**2)

DEFICIENCIES AND PROBLEMS A focal length of zero will result in a divide exception error and Straw will bomb.

REFERENCES See the Frame command for more info on frame/window size. See also LONG81, LACR83 and POTM82.

ROUTINE NAME COLOR - This defines intensity, texture maps, paint tables and colors.

CALLING SEQUENCE Color: <identifier> <red> <green> <blue> <ambient coeff> <diffuse coeff> <reflection coeff> <transparency coeff> <index of refraction> <specular coeff> <glossiness>
&
<image_map identifier> <map switch> <min pix> <max pix> <min line> <max line> <u repeat> <v repeat>
/* case 1: a sphere */
<min u> <max u> <min v> <max v>
/* case 2 : planar surface */
<x(u=0,v=0)> <y(u=0,v=0)> <z(u=0,v=0)>
<x(u=1,v=0)> <y(u=1,v=0)> <z(u=1,v=0)>
<x(u=0,v=1)> <y(u=0,v=1)> <z(u=0,v=1)>
/* case 3 : bi-cubic patches */
<min u> <max u> <min v> <max v>
&
<texture_map identifier> <map switch> <min pix> <max pix> <min line> <max line> <u repeat> <v repeat> <S(Fu,Fv)> /* case 1: a sphere */
<min u> <max u> <min v> <max v>
/* case 2 : planar surface */
<x(u=0,v=0)> <y(u=0,v=0)> <z(u=0,v=0)>
<x(u=1,v=0)> <y(u=1,v=0)> <z(u=1,v=0)>
<x(u=0,v=1)> <y(u=0,v=1)> <z(u=0,v=1)>
/* case 3 : bi-cubic patches */
<min u> <max u> <min v> <max v>
&
<paint_table identifier> ;

ARGUMENTS keyword <identifier> - Name of the color to be used by a surface command.
integer (0 - ?) <red> - gives strength of red ambient light
integer (0 - ?) <green> - gives strength of green ambient light
integer (0 - ?) <blue> - gives strength of blue ambient light
real <ambient coeff> - saturation of the final color.
integer (0 - ?) <diffuse coeff> - gives degree of diffuse reflection.
real <reflection coeff> - amplification of incident light reflected.
real <transmission coeff> - amplification of incident light transmitted.
real <index of refraction> - bends light passing through an object.
integer (0 - ?) <specular coeff> - brightness of reflection.
real <glossiness> - size of reflection.
keyword '&' - required delimiter for intensity mapping.
keyword <image_map identifier> - identifier set up by Intensity_map command.
keyword <map switch> - either 'transposed' or 'normal' are used to rotate the mapping 90 degrees.
integer <min pix> - minimum pixel in mapped image.

integer <max pix> - maximum pixel in mapped image.
integer <min line> - minimum scan line in mapped image.
integer <max line> - maximum scan line in mapped image.
integer <u repeat> - map repeat in u parameter.
integer <v repeat> - map repeat in v parameter.
keyword <texture_map identifier> - identifier set up by Texture_map command.
keyword <map switch> - either 'transposed' or 'normal' are used to rotate the mapping 90 degrees.
integer <min pix> - minimum pixel in mapped image.
integer <max pix> - maximum pixel in mapped image.
integer <min line> - minimum scan line in mapped image.
integer <max line> - maximum scan line in mapped image.
integer <u repeat> - map repeat in u parameter.
integer <v repeat> - map repeat in v parameter.
real <S(Fu,Fv)> - scale of texture function derivitives Fu and Fv.
keyword '&' - required delimiter for texture mapping.
keyword <paint_table identifier> - Name of paint table given by Paint_table command.

DESCRIPTION The color command is used to set up the color intensity map, texture, map and paint table of an object. The absolute minimum required for a color to be seen is a definition for the following arguments:
<identifier>, <red>, <green>, <blue>, <ambient coeff>

The reason for the question mark in the color command is due to the limit of the users display system. A color ambience of 500 is unreasonable for a display system whose limit is 255 (at least for direct viewing). The problem is that the internal dynamic range for intensity is greater than the displayable dynamic range.

IMPLEMENTATION The parametric definition of a sphere used is:

$$X = Xo + r*\cos(u)*\cos(v)$$
$$Y = Yo + r*\sin(u)*\cos(v)$$
$$Z = Zo + r*\sin(v)$$

DEFICIENCIES AND PROBLEMS This is a hard command to use. The maximum number of colors is 100. There is a parameter for this in the file COMMON.F77. There doesn`t seem to be a provision for mapping onto quadric surfaces. Presidence is given to those operators which appear latest in the color definition. It is very diffucult to use many different techniques from the color command at once. Straw seems to allow only one type of coloring at a time. Any image trranslation must be done externally to the straw program.

REFERENCES See the Straw code, intensity map command and the texture map command and the paint_table command.

ROUTINE NAME COTOMO - converts a color (type 26) image to a monocrome (type 8).

CALLING SEQUENCE Cotomo Enter directory name:<directory>
Enter input file name or number:<infile name>
Input color image: ..
directory: ..
name: ..
number: ..
comment: ..
date: ..

.br Enter output file name: <outfile name>
Output monochrome image
directory: ..
name: ..
number: ..
comment: ..
date: ..
Enter red multiplication coefficient:
<red coef>
Enter green multiplication coefficient:
<green coef>
Enter blue multiplication coefficient:
<blue coef>

ARGUMENTS keyword <directory> - This is a DBS directory created by the Dcreate command, see U-107.
keyword <infile name> - the type 26 DBS input file you want to process.
keyword <outfile name> - the type 8 DBS output file you want to process.
real (0..1) <red coef> - .33 for unbiased, .30 for NTSC.
real (0..1) <green coef> - .33 for unbiased, .59 for NTSC.
real (0..1) <blue coef> - .33 for unbiased, .11 for NTSC.

DESCRIPTION Cotomo converts a DeAnza image in a DBS format type 26 (color) to DBS format 8 (monocrome). A type 8 format is essential to texture mapping in Straw. The user may only operate on inputs and outputs in the same DBS directory. Further the user must save the input with the Ssave command described elsewhere in this manual in order to establish a type 26 input file. The type 8 image appears only in the red channel.

IMPLEMENTATION The unbiased conversion formula is:
L = 0.33*R + 0.33*G + 0.33*B
The NTSC (National Television Standards Committee) conversion formula is:
L = 0.30*R + 0.59*G + 0.11*B
Note that all coefficients must sum to one.

DEFICIENCIES AND PROBLEMS There should be a formula to obtain the half

tone masks for printing purposes so that a color seperation would not have to be done at the printers. Infact if this software were to output on a monocrome device (versetec, tekcopier, laser printer) the color could be reconstructed. It would be nicer if the software prompted the user for biased or NTSC converstion rather than have the user look this stuff up.

ROUTINES CALLED SAVCL, OPNDR, TNOUA, TIDEC, LSTFL, TDIP, PUTDT, CLSUN, TNOU, GET$DT

REFERENCES See the Intensity_map command elsewhere in this manual. See also User bulliten 107.

ROUTINE NAME EXECUTE - Calculates an image.

CALLING SEQUENCE Execute ;

ARGUMENTS None.

DESCRIPTION Execute may be performed iterativly during a single run of Straw. This feature is what gives Straw flexibility as an animation tool. By altering the data base after an execution a reexecution will take the alterations into account when generating the next picture.

ROUTINE NAME FRAME - contours the output of Straw to a user specified size.

CALLING SEQUENCE FRAME: <r_size> <s_size> <r_pixels> <s_lines> <i_pixel> <f_pixel> <i_line> <f_line> ;

ARGUMENTS

real <r_size> - the horizontal dimensions of the camera frame to become the output screen.
real <s_size> - the vertical dimensions of the camera frame to become the output screen.
real <r_pixels> - horizonal number of pixels on the output display.
real <s_lines> - number of scan lines in the output device.
real <i_pixel> - initial pixel number.
real <f_pixel> - final pixel number.
real <i_line> - initial line number.
real <f_line> - final line number.

DESCRIPTION The frame command allows the user to generate output for various display devices. Frame allows the user to shape the Straw output picture and align it on the screen of raster devices with up to 1024 by 1024 resolution. For the purpose of debugging pictures it is often useful to set the picture resolution to 64 by 64 and then zoom into the picture. The frame commands last four arguments are optional. The frame interacts with the focal length. For <r_size> of 30 (millimeters) by <s_size> of 30 (mm)

IMPLEMENTATION The following formula give relationship between various parameters in the Frame command:
r_pixels = f_pixel - i_pixel + 1
s_lines = f_line - i_line + 1
"The basic law governing the formation through a lens can be described by lens formula used in geometrical optics:

$$1/D + 1/V = 1/F$$

where D is the object distance, V is the Image distance, and F is the focal length of the lens, all measured along the optical axis. " [POTM82] This is always true in Straw except for the post processing techniques which limit the lens aperture.

DEFICIENCIES AND PROBLEMS Frame does not presently support display devices with greater resolution than 1024 by 1024. To change this would change the performance of Straw running on systems with output devices of lower resolution. The frame and the focal length of the camera command specify the viewing pyramid. The viewing pyramid is not straight foward to establish without some thought and this make the frame command somewhat combersome for the average user.

REFERENCES See Long81 and POTM82.

ROUTINE NAME IMAGE_OUTPUT - opens a DBS image file for a Straw picture.

CALLING SEQUENCE Image_output: <directory> <file> ;

ARGUMENTS keyword <directory> - this is a DBS directory made by the DCREATE command.
keyword <file> - this is the name or number of the file in the DBS directory.

DESCRIPTION Image_output must be given before a picture may be executed. In the event that the user wishes to execute a Straw data base more than once Image_output must be called each time Straw has generated a picture.

DEFICIENCIES AND PROBLEMS This routine relies on the DBS software subsystem.

ROUTINES CALLED PUTDT, GETDT, ...

REFERENCES See User bulliten 107

ROUTINE NAME INITIALIZE - This routine will set the Straw database to the default setting.

CALLING SEQUENCE Initialize ;

ARGUMENTS None.

DESCRIPTION Initialize should always be called when the Straw is initially invoked. It sould never be called during an animation sequence unless the user plans to set all the features in the data base to something different or feels that the data base is to full to handle any more objects. There are definite reasons for the advanced Straw user to want to delete objects from the data base. Initialize is the only way to do this and the penalty is that the user must completly regenerate the data base from the ground up.

DEFICIENCIES AND PROBLEMS There seems to be no way to initialize one specific class of commands such as spheres or light sources. This would require extending the command parser and the Straw.

ROUTINE NAME INTENSITY_MAP - specifies source of a color image.

CALLING SEQUENCE Intensity_map: <identifier> <directory> <file> ;

ARGUMENTS keyword <identifier> - this is a token to be used later by the Straw. keyword <directory> - this is the name of the DBS file directory with the image file. keyword <file> - this is the name of the image file.

DESCRIPTION Intensity_map will open a color (type 26) DBS images file for input to the Straw mapping subsystem. Intensity_map only defines a token to be used later as a Color command keyword identifier. Paint tables take presidence over intensity maps and intensity maps take presidence over normal color command listed previous to intensity maps.

DEFICIENCIES AND PROBLEMS Since only a DBS type image will work here the user must use the ssave command to save an image in the proper format. The DIPS subsystem will not do this. The Ssave command is installed as part of the UNSPSOFTware so that the user may conviently use type 26 images. The DBS directories must be closed when straw opens them or an error will occur.

REFERENCES See Ssave.

ROUTINE NAME LIGHT - inserts a light into the object coordinate system.

CALLING SEQUENCE Light: <identifier> <x> <y> <z> <red> <green> <blue> <shawdow flag> <shadow coeff>;

ARGUMENTS keyword <identifier> - Allows the light source to be refered to by name.
real <x> - The position along the X axis in the object coordinate system.
real <y> - The position along the Y axis in the object coordinate system.
real <z> - The position along the Z axis in the object coordinate system.
real <red> - red intensity of the light source.
real <blue> - blue intensity of the light source.
real <green> -green intensity of the light source.
keyword <shadow flag> - 'shadow' or 'noshadow' indicates presents of shadows.
real <shadow coef> - determines shadows contrast as cast by light source.

DESCRIPTION The Light command will effect the color percieved by the user of an object in the object coordinate system. This extends the color notation currently used by defining the lighting characteristics of an environment. Only the <identifier> <x> <y> <z> <red> <green> <blue> arguments are absolutly necessary for there to be light.

DEFICIENCIES AND PROBLEMS Currently it is impossible to get an intuitive feel for the parameters in this command because:
A. An RGB color space model of light has no relevence to any color system artists study.
B. The upper and lower limitations of these parameters in Straw is not yet known.

There is a maximum of 20 light source currently permissible. There is a parameter for this in the file COMMON.F77.

REFERENCES See Long81

ROUTINE NAME PAINT_TABLE - a linear interpolation of color along a surface.

CALLING SEQUENCE Paint_table: <identifier> [<from_x> <from_y> <from_z> <from_red> <from_green> <from_blue> <to_x> <to_y> <to_z> <to_red> <to_green> <to_blue>];

ARGUMENTS keyword <identifier> - a token to be refered to by the color command.
real <from_x> - x position on a surface where the 'from' color starts.
real <from_y> - y position on a surface where the 'from' color starts.
real <from_z> - z position on a surface where the 'from' color starts.
real <from_red> - the red component of the 'from' color.
real <from_green> - the green component of the 'from' color.
real <from_blue> - the blue component of the 'from' color.
real <to_x> - x position on a surface where the 'to' color ends.
real <to_y> - y position on a surface where the 'to' color ends.
real <to_z> - z position on a surface where the 'to' color ends.
real <to_red> - the red component of the 'to' color.
real <to_green> - the green component of the 'to' color.
real <to_blue> - the blue component of the 'to' color.

DESCRIPTION Paint_table allows a range of colors to show up which the user does not directly define. The color limits are actually 3-dimensional vectors which define a point in color space. Paint tables take presidence over intensity maps.

IMPLEMENTATION To go from one point in color space to another a linear interpolation is used.

DEFICIENCIES AND PROBLEMS As with all rgb manipulations, the additive synthesis color theory is not well understood. A better approach might be a color naming system which allows the user various color names and modifiers. If the user tries to make the from point and the too point the same value the straw will attempt to divide by zero at run time and bomb out.

ROUTINE NAME PATCH - defines a bicubic surface

CALLING SEQUENCE patch : <identifier> <color> <r_min> <list of control points> ;

ARGUMENTS keyword <identifier> - a means of labeling the patch which is not used.
keyword <color> - a color name for the patch color.
real <r_min> - the smaller r_min gives better patch resolution.
keyword <list of control points> - 16 points defined by Vertex.

DESCRIPTION In development...

ROUTINE NAME PATCH_TREES - reads a DBS file containing patches.

CALLING SEQUENCE patch_trees : <identifier> <color> <directory> <file> ;

ARGUMENTS keyword <identifier> - Names the tree, not used at this time.
keyword <color> - Defined by the Color command, gives color to patches.
keyword <directory> - A dbs directory.
keyword <file> - Name or number of file in the directory.

DESCRIPTION Patch_trees reads predefined sheets of bicubic patches and/or quadtrees of bounding volumes from the specified file. <directory> and <file> default to '*' if unspecified. '*' usually lists the files in the directory. The type of DBS file which a PATCH_TREES command can read is 1021.

REFERENCES See the section on DBS.

ROUTINE NAME PLANE - places a plane based on verticies in the Straw data base.

CALLING SEQUENCE plane : <identifier> <color> <vertex list> {& <vertex list>} ;

ARGUMENTS keyword <identifier> - This names the plane for use in the Polygon command.
keyword <color> - This is a color name defined by the Color command.
keyword <vertex list> - a list of verticies which define the boundary.
keyword & <vertex list> - verticies which define internal holes.

DESCRIPTION

The Plane command enters the plane into the Straw data base. Nothing is displayed until the Polygon command. 1000 planes are permissible.

ROUTINE NAME POLYHEDRON - displays a goup of planes.

CALLING SEQUENCE polyhedron : <identifier> ;

ARGUMENTS keyword <identifier> - a label for the polyhedron with no present use.

DESCRIPTION

Polyhedron gives the Straw data base the definition of all the previously defined planes since the last Polyhedron command or since the last initialization of the Straw.

DEFICIENCIES AND PROBLEMS

There are no tools for manipulating the Polyhedron.

ROUTINE NAME QUADRIC_BOUNDS - Establishes boundaries for a quadric surface.

CALLING SEQUENCE

Quadric_bounds: <identifier>, <+|-> <quadric> & .... | {+|-} <quadric> ....;

ARGUMENTS keyword <identifier> - Not used label for Straw data base.
keyword <+|-> - indicates which side of the quadric surface is used for

bounding.
keyword <quadric> - a surface defined by the Quadric command.

ROUTINE NAME QUADRIC_COEFFS - defines coefficients of a quadric surface.

CALLING SEQUENCE Quadric_coeffs: <identifier>, <color>, {actual/auxilliary}, <a0>, ... <a9>;

ARGUMENTS keyword <identifier> - For later use by the 'quadric_bounds' command.
keyword <color> - This is a color name defined by the Color command.
keyword {actual/auxilliary} - auxilliary surfaces bound actual surfaces.
real <a0>, ... <a9> - coefficients of a general quadric equation.

DESCRIPTION

Spheres, ellipsoids, paraboloids and hyperboloids are all the standard shapes which can be expressed with this equation. Also possible are rich examples where the quadric 'blows up' filling the screen except in the middle where there is a smooth hole.

IMPLEMENTATION The equation for the quadric surface is:
a0 * x**2 + al * y**2 + a2 * z**2 + a3 * x * y + a4 * y * z + a5 * z * x + a6 * x + a7 * y + a8 * z + a9 = 0

DEFICIENCIES AND PROBLEMS The problem with the quadric surface is that must users have trouble getting a 'feel' for it. In response to this a 'dictionary' of shapes available for the quadric surface is supplied in the appendix.

ROUTINE NAME SPHERE - generates a sphere in the object coordinate system.

CALLING SEQUENCE Sphere: <identifier> <color> <x> <y> <z> <r> ;

ARGUMENTS keyword <identifier> - This names a sphere for later reference.
keyword <color> - This is a color name defined by the Color command.
real <x> - This is the displacement along the X axis in the object coordinate space.
real <y> - This is the displacement along the Y axis in the object coordinate space.
real <z> - This is the displacement along the Z axis in the object coordinate space.
real <r> - this is the radius of the sphere in the object coordinate space.

DESCRIPTION Sphere is a surface command which gives the user an efficient method of generating spheres. Intersection, transparency, distortion and backdrops are all common uses of the Sphere command.

DEFICIENCIES AND PROBLEMS Currently Straw supports only 150 spheres. The number of spheres can be changed by altering the file 'common.f77'.

REFERENCES See the Straw code.

ROUTINE NAME SSAVE - save a DeAnza image to a DBS file in red line, followed by green line followed by blue line packing.

CALLING SEQUENCE Ssave
Enter directory name: <directory>
Enter initial x(pixel):<initial x>
Enter initial y(pixel):<initial y>
Enter number of pixels:<number of pixels>
Enter number of lines:<number of lines>
Enter output file name:<file name>

ARGUMENTS keyword <directory> - this is a DBS directory created by the Dcreate command, see U-107.
integer (0..511) <initial x> - x in DeAnza coordinate system.
integer (0..511) <initial y> - y in DeAnza coordinate system.
integer (1..512) <number of pixels> - the number of pixels per line.
integer (1..512) <number of line> - the number of lines per screen.
keywoed <file name> - the Dbs file name you want for the image.

DESCRIPTION Ssave save a DeAnza image in a DBS format so that the red, green, and blue lines follow each other in sequence. The style of storage is known as DBS type 26. This is the only type of storage permitted for the purpose of intensity mapping. The use of this command is therefore essential for the flexible use of intensity mapping. The DeAnza coordinate system is such that the lower left corner of the screen is 0,0 and the upper left is 0,511.

ROUTINES CALLED SAVCL, OPNDR, TNOUA, TIDEC, LSTFL, TDIP, PUTDT, CLSUN

REFERENCES See the Intensity_map command elsewhere in the manual. See also User bulliten 107.

ROUTINE NAME TEXTURE_MAP - specifies source of a monocrome image.

CALLING SEQUENCE Texture_map: <identifier> <directory> <file> ;

ARGUMENTS keyword <identifier> - this is a token to be used later by the Straw.
keyword <directory> - this is the name of the DBS file directory with the image file.
keyword <file> - this is the name of the image file.

DESCRIPTION Texture_map will open a monocrome (type 8) DBS images file for input to the Straw mapping subsystem. Texture_map only defines a token to be used later as a Color command keyword identifier.

DEFICIENCIES AND PROBLEMS Since only a DBS type 8 image will work here the user must use the Cotomo command to save an image in the proper format. The DIPS subsystem will not do this. The Cotomo command is installed as part of the UNSPSOFTware so that the user may conviently use type 8 images. The DBS directory must be closed when opened by the texture_map command otherwise an error will occur.

REFERENCES See Cotomo.

ROUTINE NAME VERTEX - establishes the coordinates of a vertex.

CALLING SEQUENCE vertex : <identifier> <x> <y> <z> ;

ARGUMENTS keyword <identifier> - the name of the vertex to be used later.
real <x> - world coordinate x position.
real <y> - world coordinate y position.
real <z> - world coordinate z position.

DESCRIPTION

The vertex command is providing information for the Straw data base. Planes and patches are built out of verticies.

In addition to generating a raster image with a pin-hole camera model, Straw can optionally save information about each image sample point. These sample points may be converted into an actual raster image by a post-processor.

A post-processor, BLUR, adds motion blur [5][6][7], due to a finite exposure time of real cameras, to moving surfaces. This is accomplished by convolving all image samples which belong to a moving surface with a point-spread function(PSF) [8] computed from the path and velocity of the motion, and the duration of the exposure. This convolution can be performed equally well in the spatial or the frequency domain.

The following paragraphs describe a procedure for modeling motion blur in computer-generated images by the use of BLUR.

(A) Introduction to Motion Blur

The removal of camera degradation to recover the original image based on some a priori knowledge of the degradation phenomenon is called image restoration [8]. In synthesizing motion blur the problem is almost inversed, that is, the objective is to generate an appropriate degradation function given an idealized description of the scene.

There are two principal reasons for motion blur:

1. Movements of Objects:
The motion of objects in the scene is the most common cause for image blurring.

2. Movements of the Shutter:
Film is exposed in a camera by the movement of the shutter across the film plane.

The generation of motion blur in computer-synthesized images consists of two stages:
1) a hidden-surface program generates intensity sample points of an instantaneous image identifying points which in motion and giving the image path of the projected motion;
2) BLUR which blurs the moving points by convolving them with the optical system-transfer functions [8] derived from the image path and merges them with the stationary points into a final raster image. Where the optical system-transfer function for uniform motion blur is a sinc function.

The process of the addition of motion blur is shown in Figure 1. First, separate sample points of a moving object with the same path r'(t) into a raster image f. The motion blur PSF h is computed from the object path r'(t), the exposure time t:frame, and the exposure length T:frame as a raster image h. The images f and h are then convolved into a blurred image f*h. This convolution can be performed either directly in the spatial domain, or optionally images f and h can be converted by FFT into F and H, respectively, in the frequency domain, multiplied into FxH, and then converted back into f*h by an inverse FFT. Finally, all blurred images of the moving objects are merged with the image of the stationary objects into the output raster image.

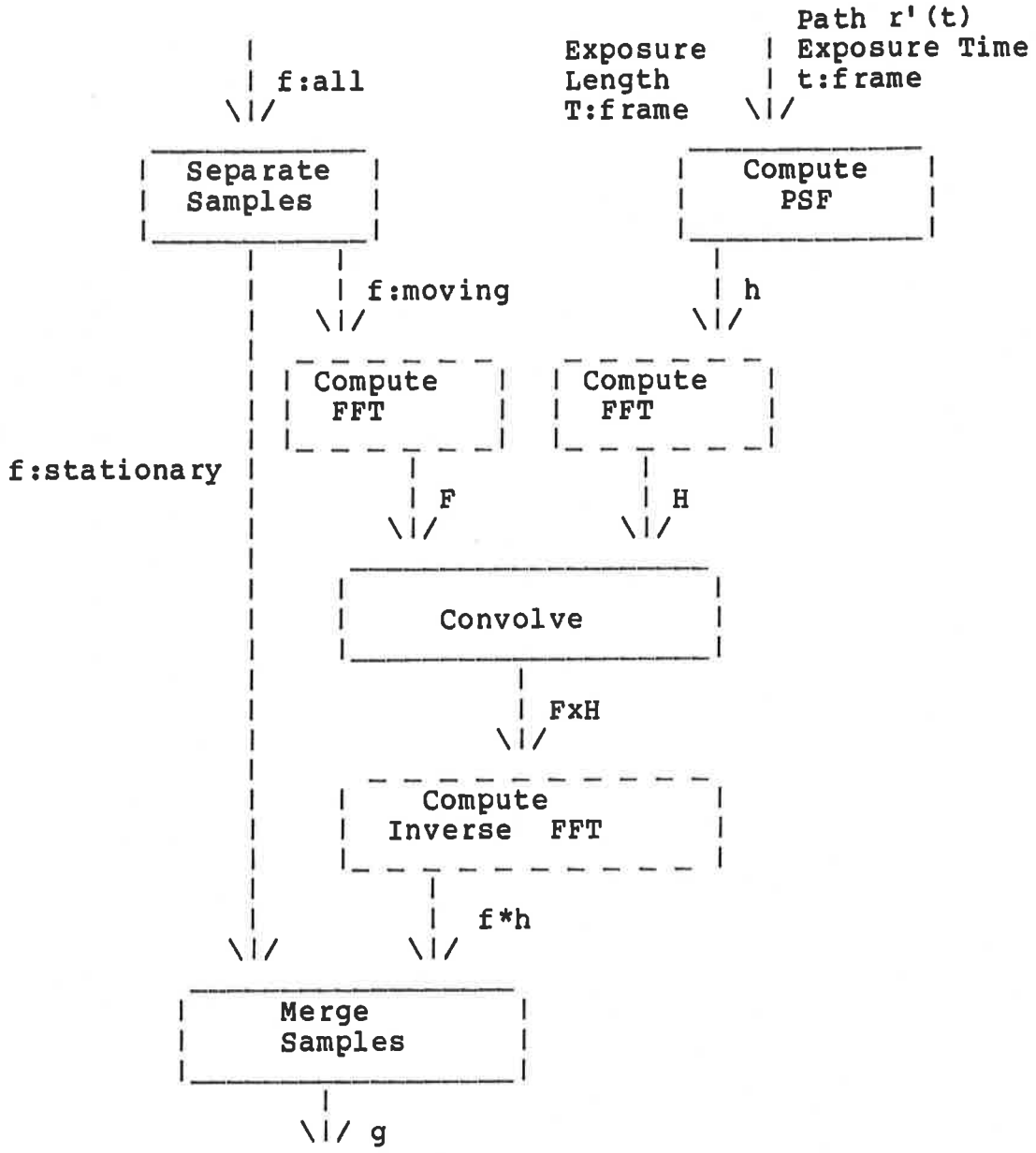


Figure 1 Block diagram of the motion-blur processor

(B) Processor BLUR

BLUR consists of nine sub-routines as following for generating motion blur described in Figure 1.

1. CIR : Convert see-format color image to real color format
2. CRI : Convert real color image format to see-format
3. FFT2D : FFT transform of real color image
4. MFFT2D: Convert color FFT image to image format
5. FBLUR : Convolve motion blur operator in frequency domain
6. SPLIT : Convert a comb-sample [6] image to real images
7. SBLUR : Convolve motion blur operator in spatial domain
8. IFFT2D: Inverse FFT transform of real color image
9. MERGE : Merge real color images into an see-format image

Figure 2 shows you the images type before and after applying a particular routine.

Figure 3 shows you the typical paths to generating motion blur.

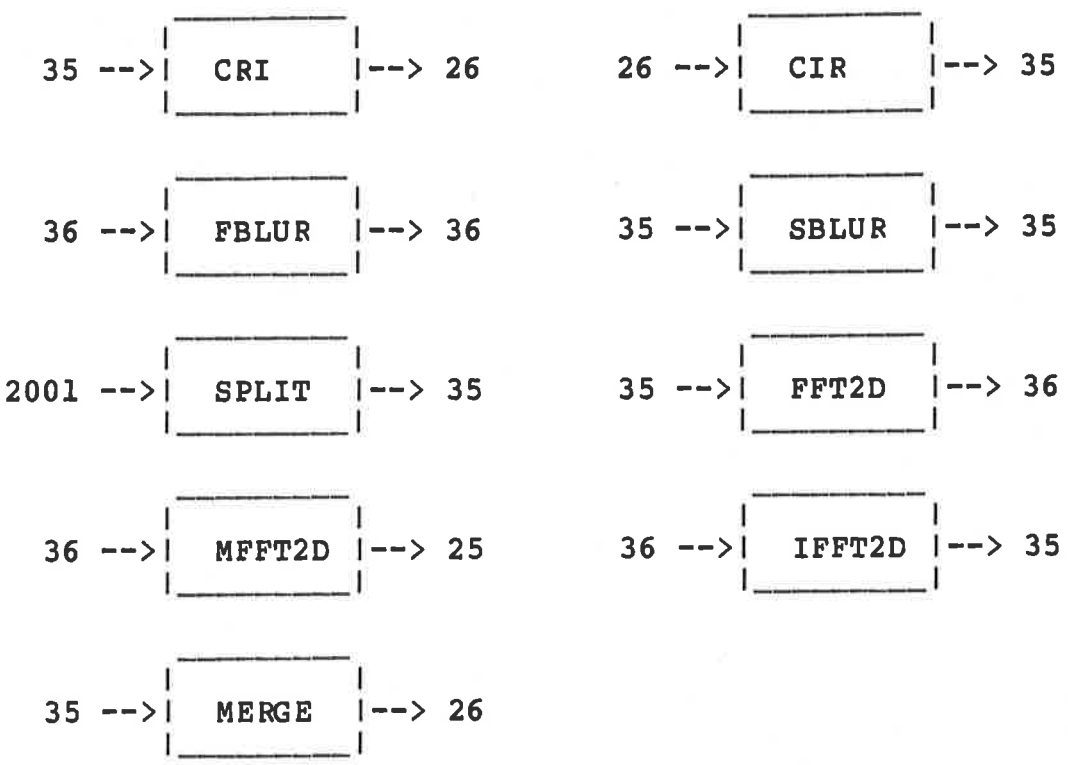


Figure 2 Image type before and after applying an operator

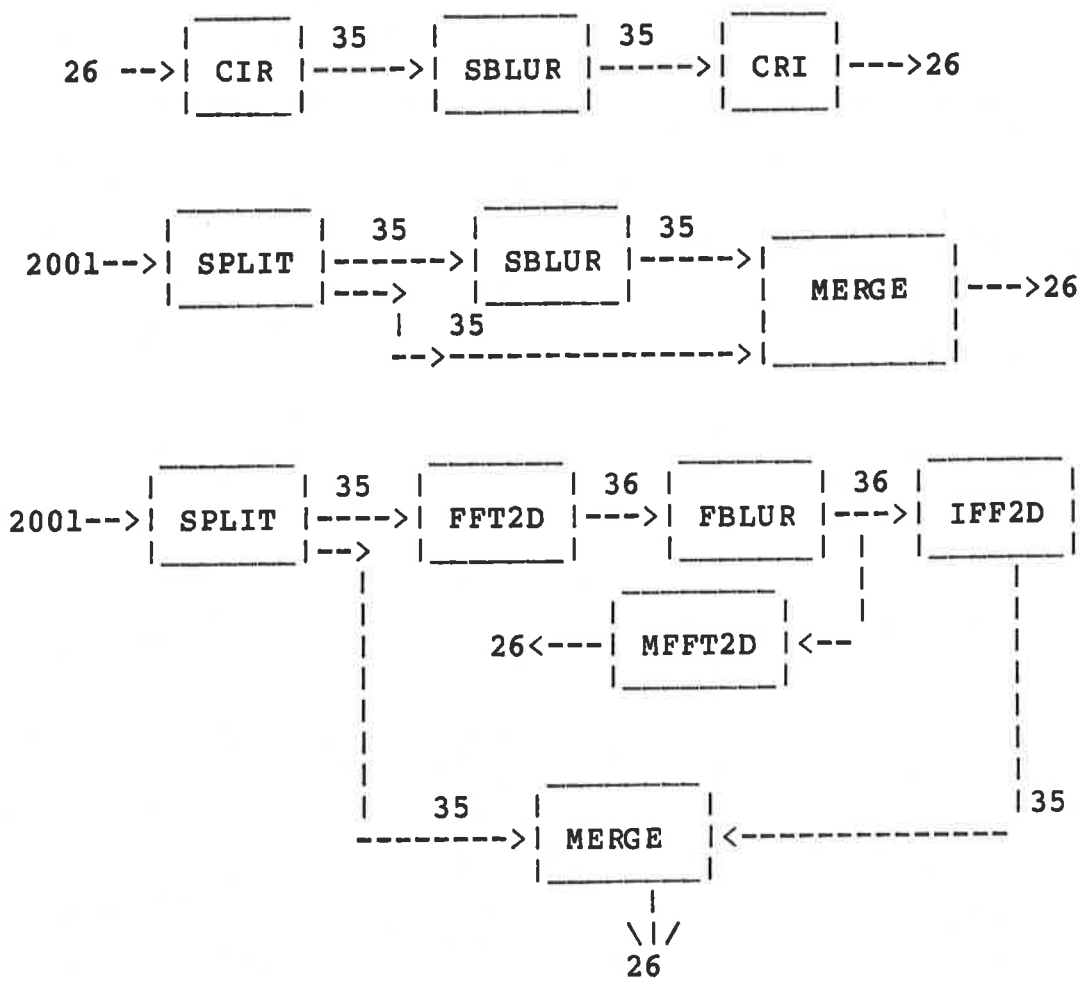


Figure 3 Typical paths to generating motion blur

(C) Examples and Comments

In this segment, we will show you how to invoke sub-routines of BLUR. Figure 4, the unblurred image, is the original image be used in the following examples. Underscored characters are typed in by user. The key sub-routines are SBLUR and FBLUR.

1. CIR: Type 26 generated by STRAW is the color output type and the mapped image input type. Type 35 is the real image type.

2. CRI: Same as CIR except the input and output image types are exchanged. The following is the procedure invoking CRI. CIR has same process.

```
OK,SEG STRAW>BLUR>CRI
Enter directory name:IMAGES
Enter input file name or number:26
Input real color image:
  directory: IMAGES
  name:      SBLUR D2 S0.2 111 111
  number:       26
  comment:   BLURRED REAL COLOR IMAGE
  date:      FRI, 09 NOV 1984   23:28:15
Enter output file name:CRI D2.0 S0.2 111 111 2
Output real image:
  directory: IMAGES
  name:      CRI D2.0 S0.2 111 111 2
  number:       28
  comment:   CONVERTED FROM REAL COLOR IMAGE
  date:      FRI, 09 NOV 1984   23:32:11
```

3. SPLIT: Type 2001 is the output image type of POINT_OUTPUT of STRAW. SPLIT is used to separate the moving objects from stationary image. The output images have four bands. Adding the fourth band (to the red, green, and blue intensity bands) is required for computing in each blurred image the fraction of the exposure length T:frame that the

moving object overlaps each pixel. This band contains the exposure length T:frame in every pixel that the instantaneous image of the moving object overlaps and 0 elsewhere.

```
OK, SEG STRAW>BLUR>SPLIT
Enter directory name:images
END   DIRECTORY <IMAGE1                                    >
Enter z-buffer file name or number:62
 Input z-buffer file:
  directory: IMAGES
  name:      PO_OUT
  number:       62
  comment:   WRITTEN BY: STRAW
  date:      TUE, 30 OCT 1984   12:21:48
 Z-buffer input file:
          number of parameters:     129
          number of samples:       7892
          number of buffers:          8
          number of scan lines:      64
          number of pixels:          64
 Enter number of moving images:
{1}
 Enter number of moving surfaces in image  1:
{1}
 Enter surface identification range:
20 40
20 40
Enter output file name:SPLIT IMAGES
Output real image:
  directory: IMAGES
  name:      SPLIT IMAGE1
  number:         0
  comment:   WRITTEN BY: SPLIT
  date:      WED, 05 DEC 1984   01:10:50
   Band 1 completed ....
   Band 2 completed ....
   Band 3 completed ....
   Band 4 completed ....
Enter output file name:SPLIT STATIONARY IMAGE
Output real image:
  directory: IMAGES
  name:      SPLIT STATIONARY IMAGE
  number:         1
  comment:   WRITTEN BY: SPLIT
  date:      WED, 05 DEC 1984   01:11:34
   Band 1 completed ....
   Band 2 completed ....
   Band 3 completed ....
   Band 4 completed ....
```

4. FFT2D: Process same as CIR. But, be careful of input image type.

5. IFFT2D: Process same as FFT2D. But, images type are inversed.

6. MFFT2D: Process same as IFFT2D. Figure 5 shows the output of MFFT2D, where the input is the image in Figure 4.

7. SBLUR: SBLUR is the most tricky sub-routine of BLUR. SBLUR is performed directly in the spatial domain. We show you the execution process, then describe the parameters used.

```
OK,SBLUR
Enter directory name:IMAGES
Enter input file name or number:3
Input real color image:
  directory: IMAGES
  name:      CIR 26 TO 35
  number:        3
  comment:   CONVERTED FROM COLOR IMAGE
  date:      FRI, 09 NOV 1984   22:03:35
Enter output file name:SBLUR D2 S0.2 111 111 4
Output blurred image
  directory: IMAGES
  name:      SBLUR D2 S0.2 111 111 4
  number:       26
  comment:   BLURRED REAL COLOR IMAGE
  date:      FRI, 09 NOV 1984   23:28:15
Enter duration time and snap time:
2, 0.2
Enter coefficients of motion in u:
1 1 1
Enter coefficients of motion in v:
1 1 1
Enter number of path steps:
2

 Contents of the blur operator:
      i      u      v    coeff
      1      0      0    0.500
      2      3      3    0.500
   Band 1 completed ....
   Band 2 completed ....
   Band 3 completed ....
```

The blurred image obtained by applying the above process is shown in Figure 6.

Exposure time is the dominant factor for motion blur. The exposure time would be increased with a corresponding increase in the motion blur.

Snap time doesn't have as much of an effect on motion blur as exposure time. It provides the information to determine the amount of time T:frame that each pixel in the stationary image is visible. The value of snap time must be smaller than the value of exposure time.

Coefficients of u and v determine the path of the moving objects. Figure 7 shows the blurred image by setting u:=(.1 .1 .1) and v:=(1 1 1). v affects the vertical direction for the motion blur and u, horizontal. First coefficient of u or v is the least significant factor and the last is the most.

The effects of the number of path steps can be seen by comparing Figure 6 and Figure 8. In Figure 6, we set it 2 and set it 6 in Figure 8. The number of path steps determine the number of aliasing objects shown in blurred image. It is used to simulate multiple exposure.

8. FBLUR: Same as SBLUR. But, it is applied in the frequency domain and doesn't have snap time and path steps effects. You can see the differences between Figure 8 and Figure 9 which are obtained by applying SBLUR and FBLUR respectively. The execution process is shown below.

```
OK,SEG STRAW>BLUR>FBLUR
Enter directory name:IMAGES
Enter input file name or number:30
Input real color image:
  directory: IMAGES
```

```
  name:       FFT2D 35 TO 36
  number:        30
  comment:    FFT OF REAL COLOR IMAGE
  date:       THU, 15 NOV 1984   20:31:15
Enter output file name:FBLUR t4 .1.1.1 222
Output blurred image
  directory: IMAGES
  name:       FBLUR T4  .1.1.1 222
  number:        5
  comment:    BLURRED REAL COLOR IMAGE
  date:       WED, 05 DEC 1984   00:52:42
Enter time exposure:
4
Enter coefficients of motion in u:
.1 .1 .1
Enter coefficients of motion in v:
2 2 2
    Band 1 completed ....
    Band 2 completed ....
    Band 3 completed ....
```

9. MERGE: Up to now I have not made this routine work yet. I traced the source program and found that MERGE tried to open a same image input file repeatedly and assign four different file units to the same file for attaching to four bands simultaneously. Primos always stopped the process when MERGE tried to open a file twice, with the Primos prompt 'file in use'. Maybe the previous version of Primos accepts this kind file handling.

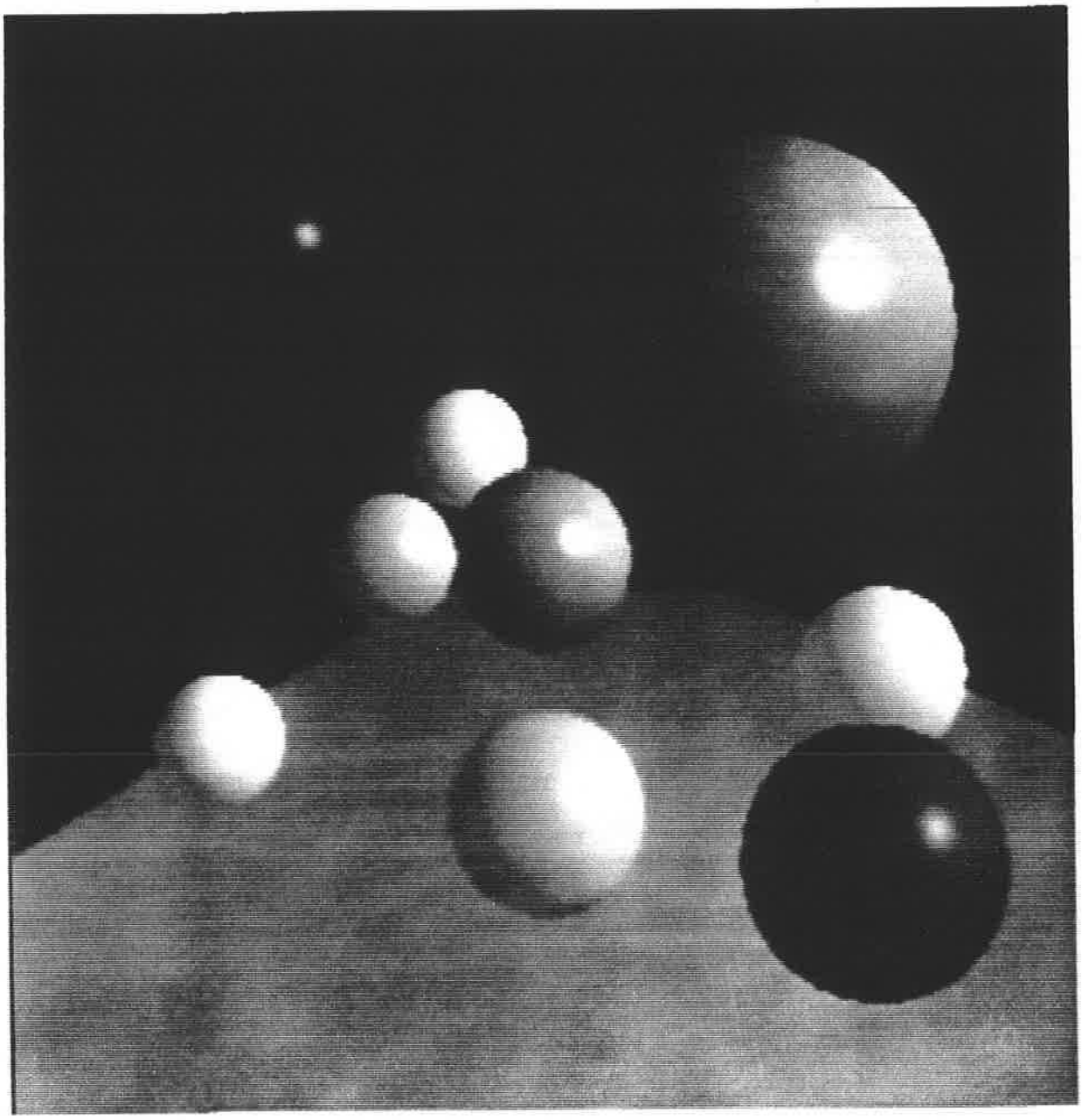


Figure 4 Unblurred image

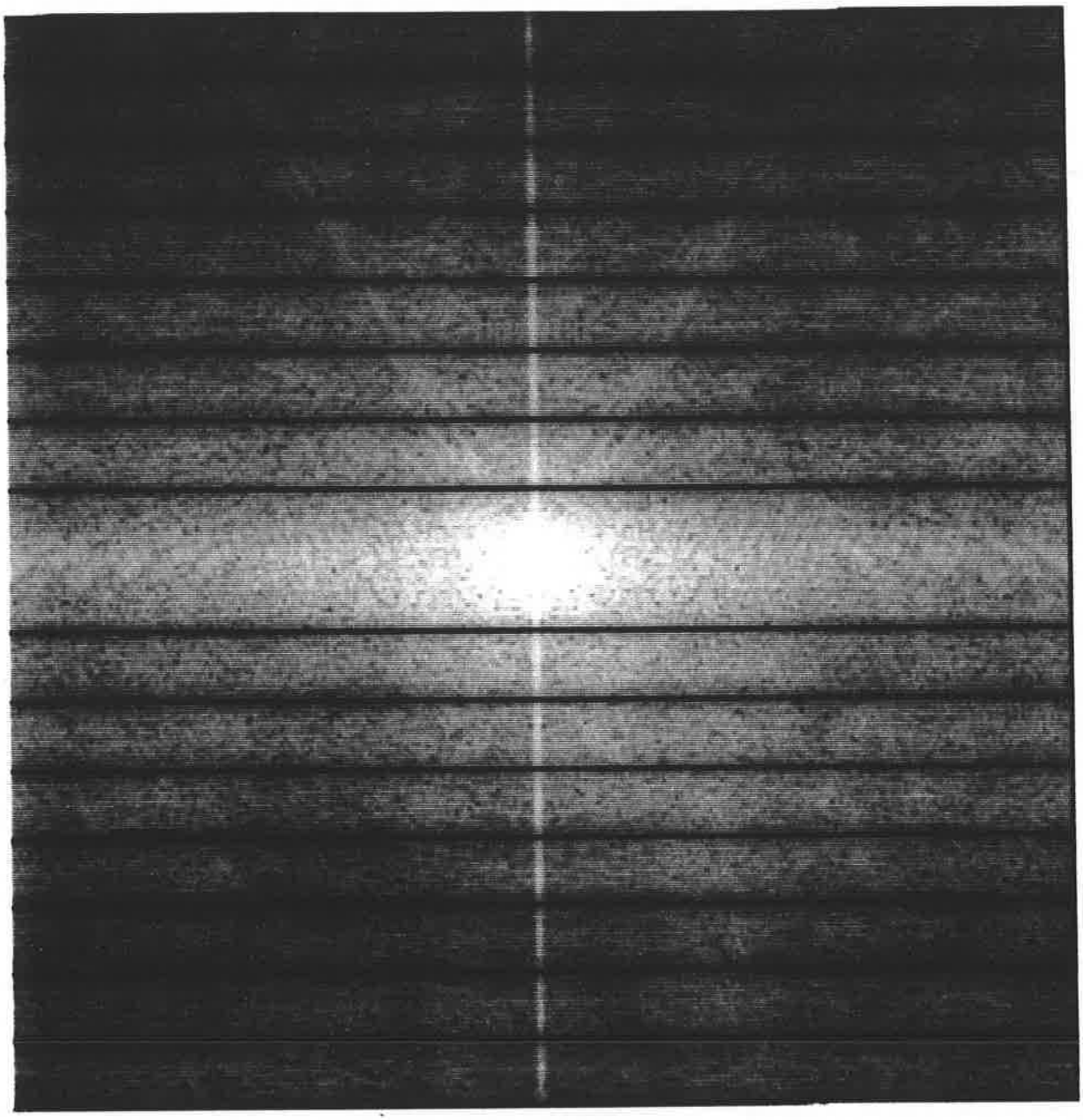


Figure 5 Output image of MFFT2D(Fourier transform)

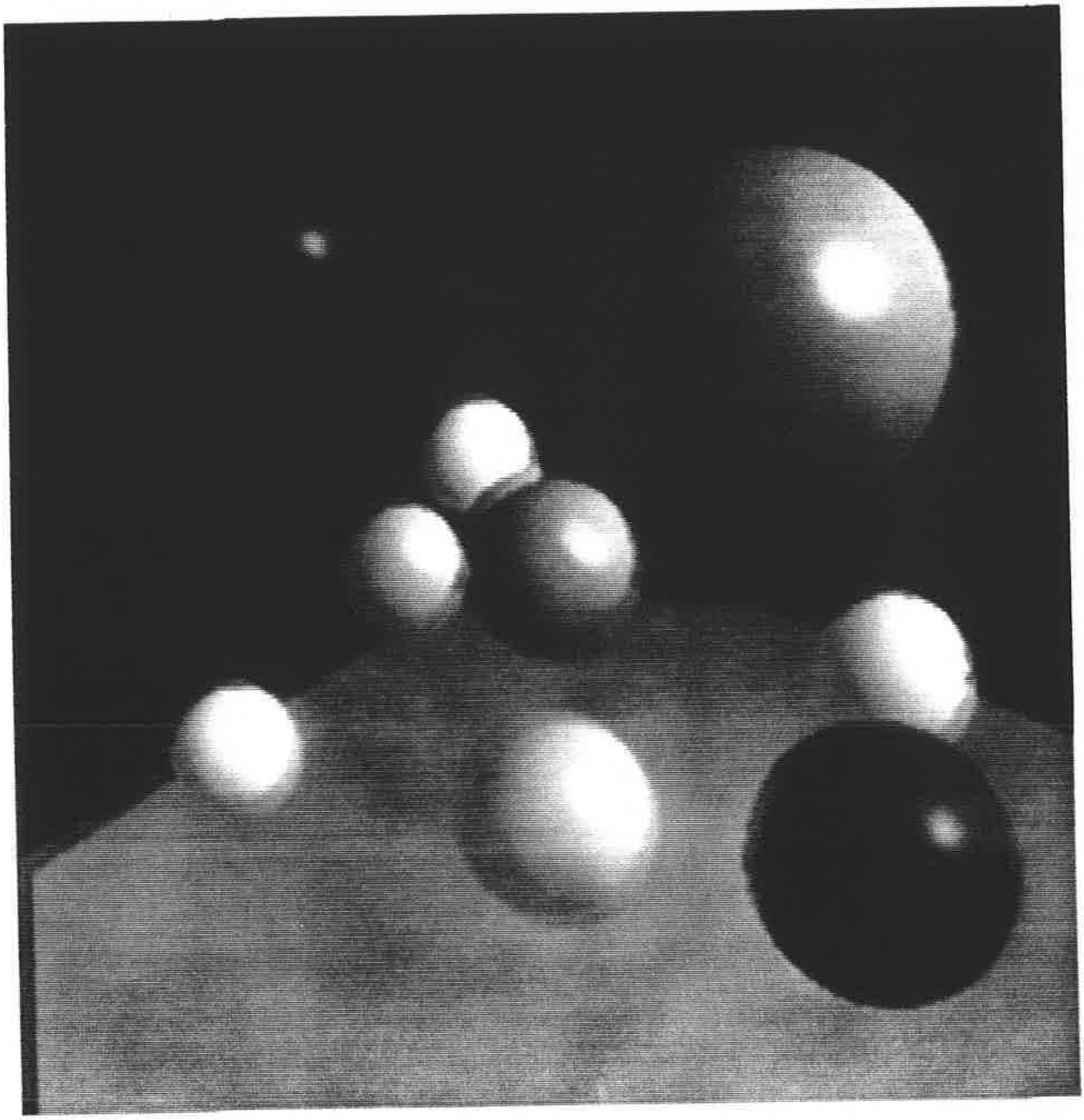


Figure 6 Blurred image (SBLUR D=2 S=.2 u=(1 1 1) v=(1 1 1) r=2)

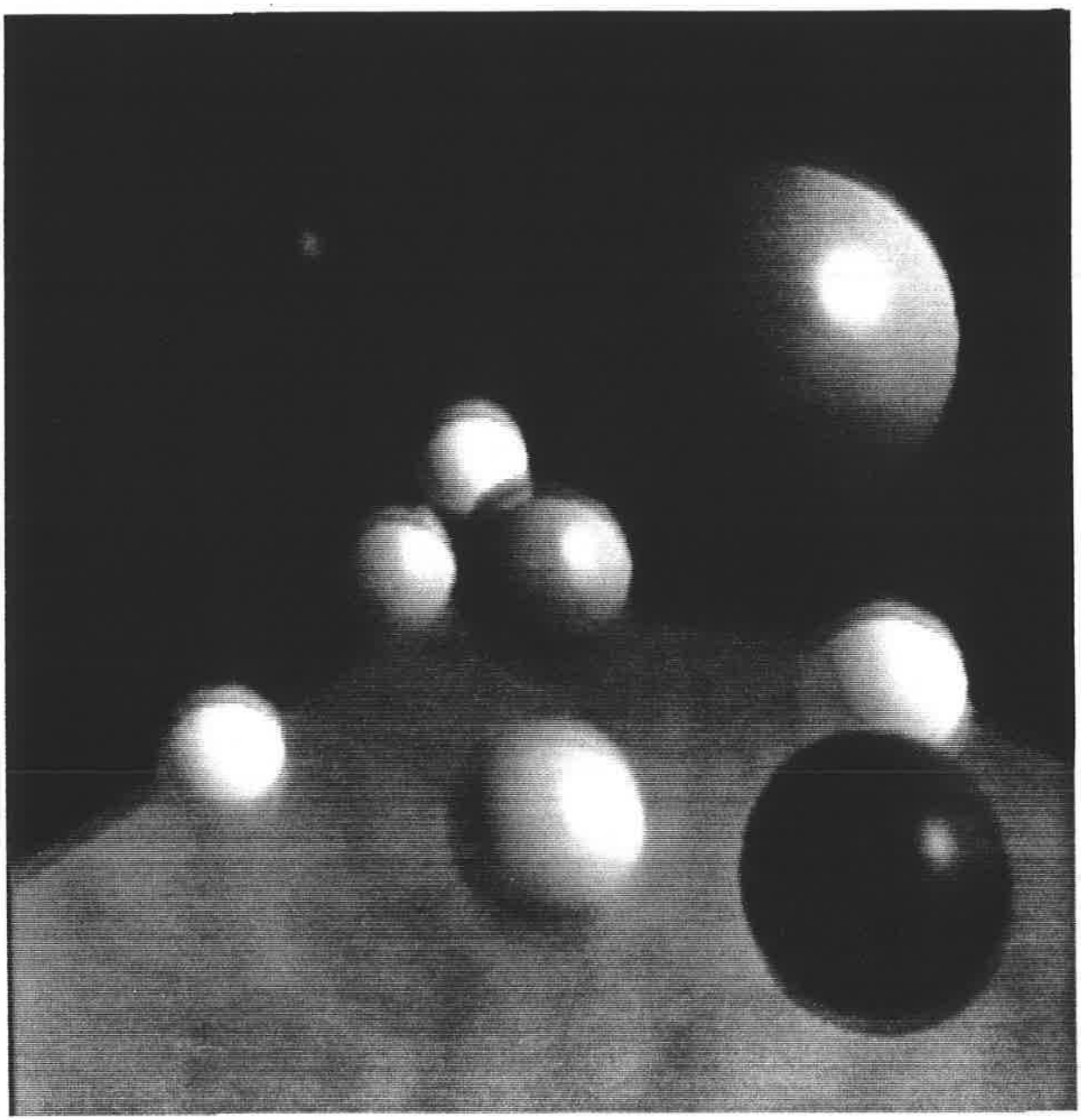


Figure 7 Blurred image (SBLUR D=2 S=.2 u=(.1 .1 .1) v=(1 1 1) r=4)

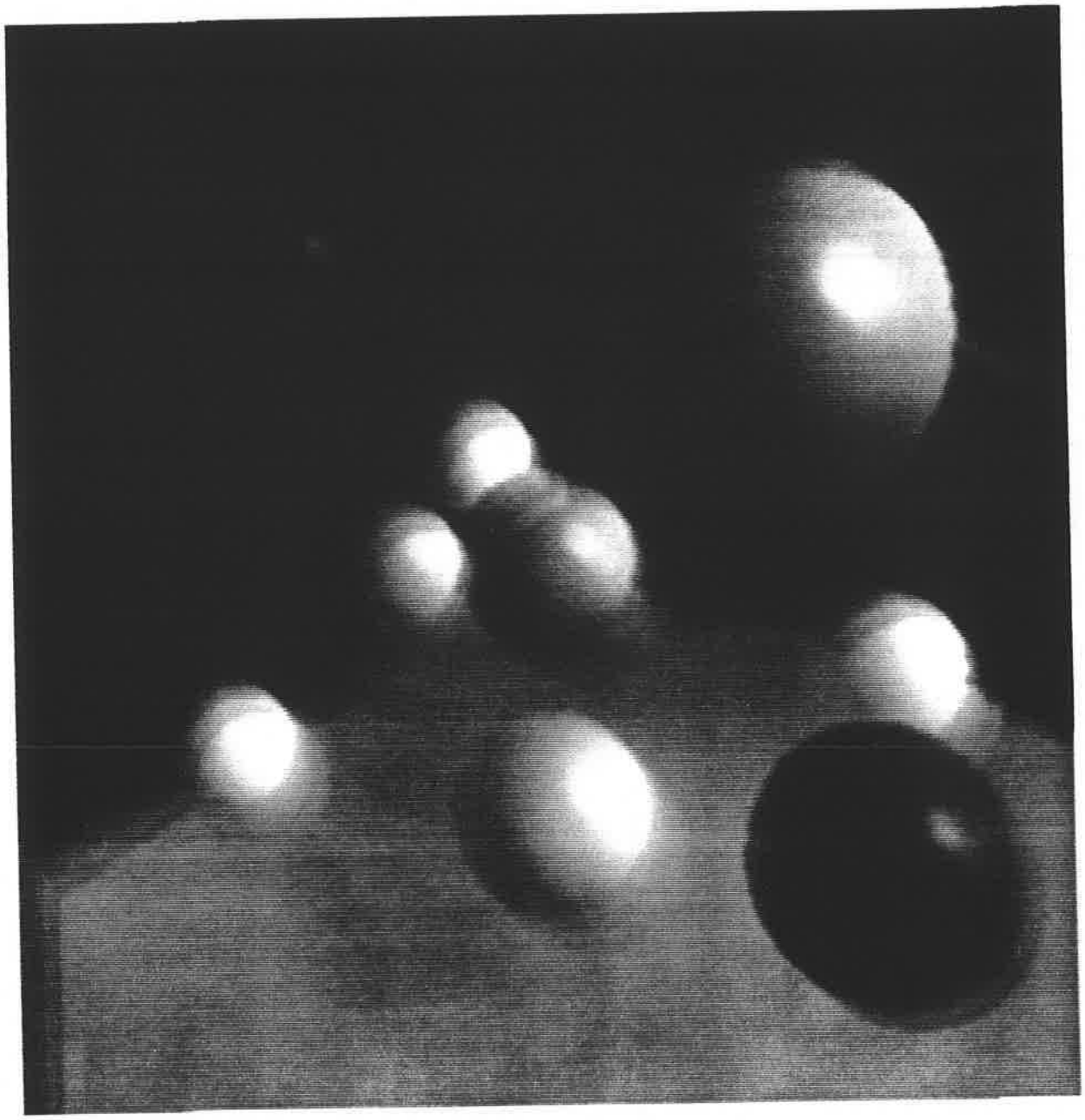


Figure 8 Blurred image (SBLUR D=2 S=.2 u=(1 1 1) v=(1 1 1) r=6)

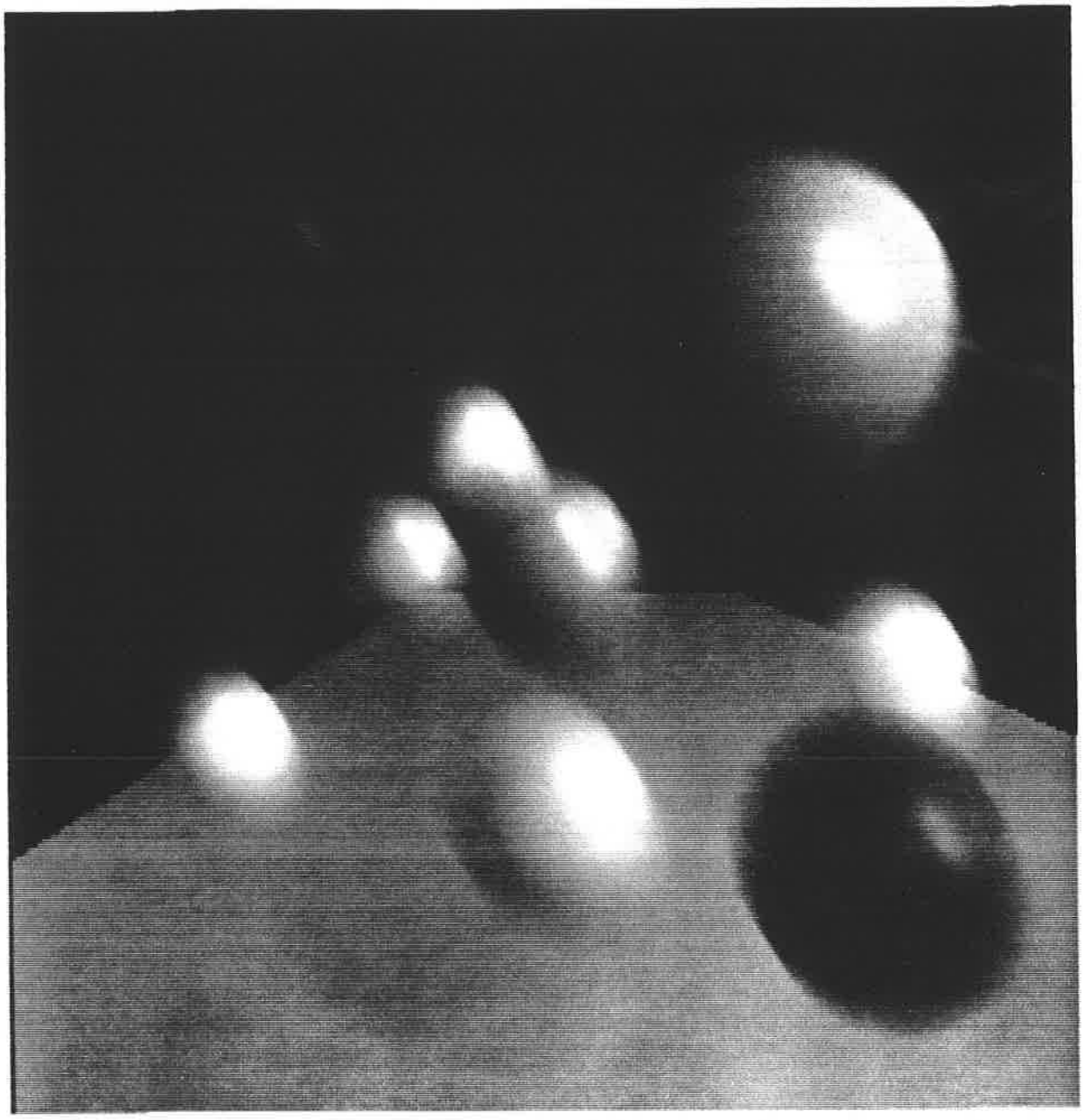


Figure 9 Blurred image (FBLUR D=4 u=(.1 .1 .1) v=(2 2 2))

```
[ Establish your output file.

[ IMAGE_OUTPUT:<directory> <file> ;

image_output:  temp_images ex_1.1;
[ The new naming convention will be as follows:
[ "ex_" indicates the input is part of the example series.
[ "G.R" where G and R are integers => 1 indicates the geometry
[ stages and the revision stages respectivly.

[ Notice that lines may be as long as you wish but are terminated by
[ a semicolon.
[ Notice that arguments need be seperated only by a space and may
[ NOT have any spaces internal to them.  "Scratch images" is
[ an invalid argument.  "Scratch_images" is an acceptable argument.
[ Upper or lower case letters are interchangeable.
[

[
[ color command.  Color is the most complex command to use.
[ COLOR:  <identifier> <red> <green> <blue> <ambient coeff> ;

color:  white 255.  255 255 1.0 ;
[ Notice that the decimal point is optional.

[ The sphere is the simplest object to use.
[ SPHERE :  <identifier> <color> <x> <y> <z> <radius>;

sphere :  ball white 0 0 0 20 ;
[ Light is important.  In order to view an object there must be  light.
[ LIGHT :  <identifier> <x> <y> <z> <red> <green> <blue> ;

light :  sun 200 200 200 1 1 1 ;
[ The camera is important.  We have to identify  the  camera  which  is
taking a
[ picture of our sphere.
[ CAMERA :  <x> <y> <z> <rot_x> <rot_y> <rot_z> <focal_length> ;

camera :  0 100 0 0 0 0 50 ;
[ The frame defines our resolution.  If we choose a small frame the
[ program will be done quickly.
[ FRAME :  <r_size> <s_size> <r_pixels> <s_lines> ...;

frame :  30 30 512 512 ;
[ this is a 512 by 512 frame

[ Execute is important.  Without Straw will do nothing.

execute;
[ Exit is the 'nice' way to terminate the Straw.  A not nice
[ way would be to allow the Straw to hit the end of the input file
[ unexpectedly, this would generate a fatal error.

exit;
```

```
[ Establish your output file.

[ IMAGE_OUTPUT:<directory> <file> ;

image_output:  temp_images ex_1.2;

[ Notice that lines may be as long as you wish but are terminated by
[ a semicolon.
[ Notice that arguments need be seperated only by a space and may
[ NOT have any spaces internal to them.  "Scratch images" is
[ an invalid argument.  "Scratch_images" is an acceptable argument.
[ Upper or lower case letters are interchangeable.
[
[ Color command.  Color is the most complex command to use.
[ Since the primaries for the subtractive colors
[ are cyan, magenta, and yellow the red, green, and blue
[ additive synthesis color theories of Straw are particularly
[ difficult to work with.  Here then are some examples
[ to get you started:
[The following colors will vary according to brightness.
[include the file in your straw code for use.
[color:  <id> r g b abient diff ref xmit ref spec glos;
color:  blue 40 210 250 .5 150 0.  0.  1.  120 50;
color:  purple 140 30 155 .5 150 0 0 1 120 50;
color:  orange1 160 40 40 .5 150 0 0 1 120 50;
color:  orange2 240 120 0 .5 130 0 0 1 1;
color:  mirror 255 255 255 .15 240 1 0 1 120 50;
color:  mirror1 10 10 10 .1 150 1 0 0 120 50;
color:  mirror2 10 10 10 .2 150 2 0 0 120 50;
color:  mirror3 10 10 10 .3 150 3 0 0 120 50;
color:  split 0 0 0 .1 150 .5 .5 0 120 50;
color:  puke_green 100 100 0 .5 150 0 0 1 120 50;
color:  yellow 300 300 50 .5 150 0 0 1 120 50;
color:  crystal 0 0 0 0 0 1 1 1.1 200 50;
color:  white 300 300 300 .5 150 0.0 0.0 1.0 120 50;
[ Be careful not to give the Straw two colors with the same name.

[ Notice that the decimal point is optional.

[ The sphere is the simplest object to use.
[ SPHERE :  <identifier> <color> <x> <y> <z> <radius>;

sphere :  ball orange1 0 0 0 10 ;
[ Light is important.  In order to view an object there must be  light.
[ LIGHT :  <identifier> <x> <y> <z> <red> <green> <blue> ;

light :  sun 200 200 200 1 1 1 ;

light :  sun2 -200 200 200 1 1 1 ;


[ The camera is important.  We have to identify  the  camera  which  is
taking a
[ picture of our sphere.
```

```
[ CAMERA :  <x> <y> <z> <rot_x> <rot_y> <rot_z> <focal_length> ;

camera :  0 100 0 0 0 0 50 ;
[ The frame defines our resolution.  If we choose a small frame the
[ program will be done quickly.
[ FRAME :  <r_size> <s_size> <r_pixels> <s_lines> ...;

frame :  30 30 256 256 ;
[ this is a 256 by 256 frame

[ Execute is important.  Without Straw will do nothing.

execute;
[ Exit is the 'nice' way to terminate the Straw.  A not nice
[ way would be to allow the Straw to hit the end of the input file
[ unexpectedly, this would generate a fatal error.

exit;
```

```
[ Establish your output file.

[ IMAGE_OUTPUT:<directory> <file> ;

image_output:  temp_images ex_2.1;
[
[The following colors will vary according to brightness.
[include the file in your straw code for use.
[color:  <id> r g b abient diff ref xmit ref spec glos;
color:  blue 40 210 250 .5 150 0.  0.  1.  120 50;
color:  purple 140 30 155 .5 150 0 0 1 120 50;
color:  orange1 160 40 40 .5 150 0 0 1 120 50;
color:  orange2 240 120 0 .5 130 0 0 1 1;
color:  mirror 255 255 255 .15 240 1 0 1 120 50;
color:  mirror1 10 10 10 .1 150 1 0 0 120 50;
color:  mirror2 10 10 10 .2 150 2 0 0 120 50;
color:  mirror3 10 10 10 .3 150 3 0 0 120 50;
color:  split 0 0 0 .1 150 .5 .5 0 120 50;
color:  puke_green 100 100 0 .5 150 0 0 1 120 50;
color:  yellow 300 300 50 .5 150 0 0 1 120 50;
color:  crystal 0 0 0 0 0 1 1 1.1 200 50;
color:  white 300 300 300 .5 150 0.0 0.0 1.0 120 50;
[ Be careful not to give the Straw two colors with the same name.

[ Notice that the decimal point is optional.

[ The sphere is the simplest object to use.
[ SPHERE :  <identifier> <color> <x> <y> <z> <radius>;

sphere :  ball orange1 0 0 0 10 ;

sphere :  sky blue 0 0 0 10000 ;
[ Light is important.  In order to view an object there must be  light.
[ LIGHT :  <identifier> <x> <y> <z> <red> <green> <blue> ;

light :  moon 200 200 200 .4 .4 .4 ;

light :  star -200 200 200 .1 .1 .1 ;


[ The camera is important.  We have to identify  the  camera  which  is
taking a
[ picture of our sphere.
[ CAMERA :  <x> <y> <z> <rot_x> <rot_y> <rot_z> <focal_length> ;

camera :  0 100 0 0 0 0 50 ;
[ The frame defines our resolution.  If we choose a small frame the
[ program will be done quickly.
[ FRAME :  <r_size> <s_size> <r_pixels> <s_lines> ...;

frame :  30 30 64 64 ;
[ this is a 64 by 64 frame

[ Execute is important.  Without Straw will do nothing.
```

```
execute;
[ Exit is the 'nice' way to terminate the Straw.  A not nice
[ way would be to allow the Straw to hit the end of the input file
[ unexpectedly, this would generate a fatal error.

exit;
```

```
[ Establish your output file.

[ IMAGE_OUTPUT:<directory> <file> ;

image_output:  temp_images ex_2.2;
[
[include the file in your straw code for use.
[color:  <id> r g b abient diff ref xmit ref spec glos;
color:  blue 40 210 250 .5 150 0.  0.  1.  120 50;
color:  purple 140 30 155 .5 150 0 0 1 120 50;
color:  orange1 160 40 40 .5 150 0 0 1 120 50;
color:  orange2 240 120 0 .5 130 0 0 1 1;
color:  mirror 255 255 255 .15 240 1 0 1 120 50;
color:  mirror1 10 10 10 .1 150 1 0 0 120 50;
color:  mirror2 10 10 10 .2 150 2 0 0 120 50;
color:  mirror3 10 10 10 .3 150 3 0 0 120 50;
color:  split 0 0 0 .1 150 .5 .5 0 120 50;
color:  puke_green 100 100 0 .5 150 0 0 1 120 50;
color:  yellow 300 300 50 .5 150 0 0 1 120 50;
color:  crystal 0 0 0 0 0 1 1 1.1 200 50;
color:  white 300 300 300 .5 150 0.0 0.0 1.0 120 50;
[ Be careful not to give the Straw two colors with the same name.

[ Notice that the decimal point is optional.

[ The sphere is the simplest object to use.
[ SPHERE :  <identifier> <color> <x> <y> <z> <radius>;

sphere :  ball1 orange1 0 0 0 10 ;
[ a crystal dimple.

sphere :  ball2 crystal -3 3 3 5 ;

sphere :  sky blue 0 0 0 10000 ;
[ Light is important.  In order to view an object there must be  light.
[ LIGHT :  <identifier> <x> <y> <z> <red> <green> <blue> ;

light :  light 200 200 200 .8 .8 .8 ;
[ The camera is important.  We have to identify  the  camera  which  is
taking a
[ picture of our sphere.
[ CAMERA :  <x> <y> <z> <rot_x> <rot_y> <rot_z> <focal_length> ;

camera :  0 100 0 0 0 0 50 ;
[ The frame defines our resolution.  If we choose a small frame the
[ program will be done quickly.
[ FRAME :  <r_size> <s_size> <r_pixels> <s_lines> ...;

frame :  30 30 256 256 ;
[ this is a 256 by 256 frame

[ Execute is important.  Without Straw will do nothing.

execute;
```

```
[ Exit is the 'nice' way to terminate the Straw.  A not nice
[ way would be to allow the Straw to hit the end of the input file
[ unexpectedly, this would generate a fatal error.

exit;
```

```
[ big hues on abatu
[

initialize;
[
[ Camera and frame parameters:
[

camera:  0, -300, 50, 5, 180, -20, 100.00;
[[ camera:  0, 250, 0, 0, 0, 0, 5;

frame:  30.0, 30.0, 64 , 64 ;

shade:  10, 10.0, 10.0, 10000.0, 10000.0;
[
[ Output image file:
[

image_output:  TEMP_IMAGES, 'big hues on abatu';
[
[ Mapped images:
[ just changed this to stars....doug

intensity_map:  stars, scratch_images, 3;

intensity_map:  baby, scratch_images, 6;
[
[
[ Paint tables:
[

paint_table:  red-green

0 0 126 255 0 0

0 0 15 255 255 0

0 0 -15 255 255 0

0 0 -126 0 255 0;
[

paint_table:  green-blue

0 0 126 0 255 0

0 0 15 0 255 255

0 0 -15 0 255 255

0 0 -126 0 0 255;
[

paint_table:  blue-red
```

```
0 0 126 0 0 255

0 0 15 255 0 255

0 0 -15 255 0 255

0 0 -126 255 0 0;
[
[ Color definitions:
[

color:  hue-1 255 255 255 0.4 125.0 0.0 0.0 1.0 0.0 1

& & & red-green;

color:  hue-2 255 255 255 0.4 125.0 0.0 0.0 1.0 0.0 1

& & & green-blue;

color:  hue-3 255 255 255 0.4 125.0 0.0 0.0 1.0 0.0 1

& & & blue-red;
[ more Color definitions:
[

color:  stars 0 0 0

1.2 2.  0.0

1.0 0.0

1.0 100

&

stars normal 1 512 1 512 1 1 -3.14 3.14 -3.14 3.14

;

color:  baby 0 0 0

1.2 2.  0.0

1.0 0.0

1.0 100

&

baby normal 1 512 1 512 1 1 -3.14 3.14 -3.14 3.14

;

color:  hue-3 255 255 255 0.4 125.0 0.0 0.0 1.0 0.0 1
```

```
& & & blue-red;
color:  blue-red-mirror
0, 0, 0, 0.15, 100.0, .9, 0.0, 1.0, 250.0, 80
& & & blue-red;
color:  red-green-mirror
0, 0, 0, 0.15, 100.0, .9, 0.0, 1.0, 250.0, 80
& & & red-green;
[
[ Vertices for planes:
[
vertex:  v-9, -100, 200, 200;
vertex:  v-10, 300, 200, 200;
vertex:  v-11, 300, 200, -100;
vertex:  v-12, -100, 200, -100;
vertex:  v1 900 -400 1500;
vertex:  v2 900 -400 -1500;
vertex:  v3 -900 -400 -1500;
vertex:  v4 -900 -400 1500;
[
[ Definitions of planar surfaces:
[
plane:  bak_mirror red-green-mirror v1 v2 v3 v4;
plane:  front_mirror blue-red-mirror v-9 v-10 v-11 v-12;
polyhedron:  cube_mirror;
[
[ Spheres:
[
sphere:  universe stars 0 0 0 20000000000;
sphere:  baby_planet baby 0 0 0 20;
[
[ Light sources:
[
light:  lamp, 1000, 1000, 1000, 1.0, 1.0, 1.0, shadows, 0.0;
[
```

```
[ Execute:
[

list:  all;

status:  all;

execute;

exit;
[
```

```
[ Three sides of a cube with mapped images ....
[ ....  with a planar mirror

initialize;
[
[ Camera and frame parameters:
[ X Y Z RX RY RZ FL

camera:  0, 30, 45, 0, -90, 0, 30;

frame:  30.0, 30.0, 512, 512;
[

image_output:  image, 'outside a mandalla program mandalla';

paint_table:  red-green

0 0 126 255 0 0

0 0 15 255 255 0

0 0 -15 255 255 0

0 0 -126 0 255 0;
[
[ Color definitions:

color:  mirror 0, 0, 0, 0.15, 200.0, 1.25, 1.0, 1.0, 250.0, 80;

color:  orange 240, 120, 0, 0.40, 155.5, 0.0, 0.0, 1.0, 150.0, 1;

color:  hue-1 255 255 255 0.6 125.0 0.0 0.0 1.0 0.0 1

& & & red-green;

color:  doug 255, 255, 255, 1., 155.5, 0.0, 0.0, 1.0, 150.0, 1;

color:  purple 140 30 155 .5 150 0 0 1 120 50;
[
[ Vertices for planes:
vertex:  v-1, 0, 0, 0;
vertex:  v-2, 0, 60, 0;
vertex:  v-3, 0, 30, 90;
vertex:  v-4, 800, 0, 0;
vertex:  v-5, 800,60,0;
vertex:  v-6, 800, 30, 90;
plane:  p-1, mirror, v-1, v-4, v-6, v-3;
plane:  p-2, mirror, v-2, v-5, v-6, v-3;
plane:  p-3, mirror, v-2, v-5, v-4, v-1;
polyhedron:  prism;
[ Spheres:
[

sphere:  balll hue-1 825 30 45 30;
```

```
[ Light sources:
[

light:  lamp, 300, 300, 300, 1.0, 1.0, 1.0, shadows, 0.0;

[
[ Execute:
[

list:  all;

status:  all;

execute;

exit;
```

```
[ big hues on abatu
[

initialize;
[
[ Camera and frame parameters:
[

camera:  0, 200, 0, 0, 0, 0, 5;

frame:  30.0, 30.0, 512 , 512 ;

shade:  10, 10.0, 10.0, 10000.0, 10000.0;
[
[ Output image file:
[

image_output:  TEMP_IMAGES, 'big hues on abatu';
[
[ Mapped images:
[ just changed this to stars....doug

intensity_map:  stars, scratch_images, 3;

intensity_map:  baby, scratch_images, 6;
[
[
[ Paint tables:
[

paint_table:  red-green

0 0 126 255 0 0

0 0 15 255 255 0

0 0 -15 255 255 0

0 0 -126 0 255 0;
[

paint_table:  green-blue

0 0 126 0 255 0

0 0 15 0 255 255

0 0 -15 0 255 255

0 0 -126 0 0 255;
[

paint_table:  blue-red
```

```
0 0 126 0 0 255

0 0 15 255 0 255

0 0 -15 255 0 255

0 0 -126 255 0 0;
[
[ Color definitions:
[

color:  hue-1 255 255 255 0.4 125.0 0.0 0.0 1.0 0.0 1

& & & red-green;

color:  hue-2 255 255 255 0.4 125.0 0.0 0.0 1.0 0.0 1

& & & green-blue;

color:  hue-3 255 255 255 0.4 125.0 0.0 0.0 1.0 0.0 1

& & & blue-red;
[ more Color definitions:
[

color:  stars 0 0 0

1.2 2.  0.0

1.0 0.0

1.0 100

&

stars normal 1 512 1 512 1 1 -3.14 3.14 -3.14 3.14

;

color:  baby 0 0 0

1.2 2.  0.0

1.0 0.0

1.0 100

&

baby normal 1 512 1 512 1 1 -3.14 3.14 -3.14 3.14

;

color:  hue-3 255 255 255 0.4 125.0 0.0 0.0 1.0 0.0 1
```

```
& & & blue-red;

color:  blue-red-mirror

0, 0, 0, 0.15, 100.0, .9, 0.0, 1.0, 250.0, 80

& & & blue-red;

color:  red-green-mirror

0, 0, 0, 0.15, 100.0, .9, 0.0, 1.0, 250.0, 80

& & & red-green;
[
[ Vertices for planes:
[

vertex:  v-9, -100, 200, 200;

vertex:  v-10, 300, 200, 200;

vertex:  v-11, 300, 200, -100;

vertex:  v-12, -100, 200, -100;

vertex:  v1 900 -400 1500;

vertex:  v2 900 -400 -1500;

vertex:  v3 -900 -400 -1500;

vertex:  v4 -900 -400 1500;
[
[ Definitions of planar surfaces:
[
[[ plane:  bak_mirror red-green-mirror v1 v2 v3 v4;
[[ plane:  front_mirror blue-red-mirror v-9 v-10 v-11 v-12;
[[ polyhedron:  cube_mirror;
[
[ Spheres:
[

sphere:  universe stars 0 0 0 20000000000;

sphere:  baby_planet baby 0 0 0 200;
[
[ Light sources:
[

light:  lamp, 1000, 1000, 1000, 1.0, 1.0, 1.0, shadows, 0.0;
[
[ Execute:
[
```

```
list:  all;

status:  all;

execute;

exit;
[
```

1. Long, D. and Young, J. 1981 'STRAW' Command Processor User's Manual

2. Lacroix, V. and Vishwanathan, S. 1983 'HELP-STRAW' A User's Manual for STRAW

3. Staff of IPL, 1982 DIPS USER'S MANUAL - IPL-TR-020

4. Foley, J.D. and Van Dam, A. 1982 Fundamentals of Interactive Computer Graphics

5. Potmesil, M. and Chakravarty, I 1982 Synthetic Image Generation with a lens and Aperature Camera Model

6. Potmesil, M. and Chakravarty, I., Modeling Motion Blur in Computer-Generated Images, ACM Computer Graphics, Vol 17, no. 3, July 1983

7. Potmesil, M. 1982 Generating Three-Dimensional Surface Models of solid objects from multiple projections IPL-TR-003

8. Andrews H.C. and Hunt B.R., 1976 Digital Image Restoration

Leng-Meng Lin was born in 1957 in Keelung, Taiwan. In 1979, he received his B.S. degree in Electrical Engineering from Chung Yuan Christian University in Taiwan. Two years later, he received his M.S. degree from National Cheng Kung University in Taiwan. From 1981 to 1983, he was an techinal officer of Republic of China. Presently he is going for Ph.D. degree at Rensselaer Polytechnic Institute in Troy, NY.

Douglas Lyon is the Chief Scientist for Raytal Inc., a company involved with laser imaging in real time. He is also Chief Engineer for WRPI a college run 10000 watt radio station. Born in New York City in 1960 Doug has published in Kim one user's note, and assorted hobbiest journels. Doug posseses a B.S. from RPI and is working towards an M.E.

This is an appendix on PRIMOS. PRIMOS is an operating system which runs on Prime computers. For future versions of Straw refer to the appendix for your operating system. Straw is native to PRIMOS and this is the first operating system dependent appendix to the Straw User's Manual. The following topics should be understood, preferably in the order in which they are listed.

Editing files - In order to prepare input files for Straw the user must know how to work the editor. The recommended editor is EMACS. An excellent tutorial is available, type:
teach-emacs
Experence shows that the first time user of EMACS will fair well with only one lesson and some experienced EMACS users or a manual close at hand. Please note: to get help in EMACS type 'CTRL_'.

Displaying Images - In order to see Straw images the user must know how to work a subsystem called DIPS. See the "Dips User's Manual" (IPL-TR-020). Critical commands to learn are:
DISP - this allows a user to display an image.
S&Z - this permits scrolling and zooming.

Creating an image directory - An image directory is called a DBS directory (Data Base System). The image directory handles all data going into and out of Straw which is not text. Text data consists of a standard ASCII file which is human readable. The input text file consists of Straw commands. The output text file contains the input text plus the Straw interpretation.

To create an image directory use the Primos command DCREATE.

contains an appendix on examples. In order to get started with Straw it is advisable to copy the directory STRAW>.GETTING_STARTED into your UFD. This is done by typing:
COPY STRAW>.GETTING_STARTED <your ufd here>.STRAW
All examples are input text files. To use the input text files you must invoke Straw. Type ORGIN (to get back to your home ufd),

A *>.STRAW (to place yourself in your 'straw' environment),

PH PH_STRAW (to launch a phantom which will run the first example) The phantom will run as a background task. The system will issue a message 'phantom user is 'nn'.' where nn is an integer. This will execute a straw program listed as example 1.1 elsewhere in this manual. To find out how the phantom did when the phantom logs out you must edit the file '$print$'.
Read the *INFO* file, this will guide you through the rest of the Straw environment.
Presently, the UFD called STRAW contains a directory called .DOCUMENTATION, this is where the source for this manual may be found. A copy also exists in UNSPSOFT>DOC>STRAW.
A DBS directory need be created only once. A DBS directory has no known limit for the number of images stored (disk space has always been exceeded first).

Rolling your own Straw - The file ph_straw contains a unique file name which refers to the path name of the straw input file. This is done in

the context of a straw program. Launching this file as a phantom will therefore invoke the straw software subsytem and redirect the flow of control of the straw program to the data pointed to by the filename placed in the ph_straw file.

In summary you must edit ph_straw and put in your own file name. The convention is to use the directory called .INPUTS to store your straw input files (violations of this convention abound). Invoke dips to display the images in the DBS directory called IMAGES in order to see the straw output for the examples. The image directory is defined by the image_output command in straw.

The following information is not for the casual user and is given for completness only.

DBS is a Data Base System for supporting the Straw program (See U-107). Straw extends the current DBS types with the following types:

Type 8 is the texture image input type.
Type 26 is the color output type and the mapped image input type. It is the input to cir, dips and straw.
Type 35 is the output of split, cir and sblur it is the

input of sblur, merge and cri, fft2d
Type 36 is the fblur input fft2d, fblur output
Type 1011 is the Parametric network file type. This is the output of SURFED. This is a software subsystem for adage generated pictures.
Type 1021 is the Surface-Quadtree file type. This is read by the straw command PATCH_TREES.
Type 1022 is the Solid-Parallelepiped file type. This is input to STRAW.
Typy 1063 is a muftod image.
Type 2001 is the Z-Buffer file type split program input, it is generated by STRAW.

35 -> merge -> ? 35 -> cri -> 26 36 -> fblur -> 35 26 -> cir -> 35 36 -> mfft2d -> 25 35 -> fft2d -> 36 point_output -> 2001 2001 -> split -> 35

Camera: <x>, <y>, <z>, <rx>, <ry>, <rz>, <fl>, {perspective|orthogonal}; defines camera parameters.

Frame: <frmu>, <frmv>, <nu>, <nv>, <nuint>, <nufin>, <nvint>, <nvfin>; defines frame parameters of camera and synthetic image.

Image_output: <directory>, <file>; opens output image file.

Point_output: <directory>, <file>; opens output zbuffer file.

Paint_table: <identifier>, <min_x>, ... <max_x>, ... & <min_x>, ... <max_x>; enters a paint table.

Intensity_map: <identifier>, <directory>, <file>; opens a color image file for intensity mapping.

Texture_map: <identifier>, <directory>, <file>; opens a monochrome image file for texture mapping.

Color: <identifier>, <red>, <green>, <blue>, <c_ambient>, <c_diffuse>, <c_reflection>, <c_transmission>, <c_refraction>, <c_specular>, <k_glossiness> & <intensity map> & <texture_map> & <paint_table>; defines properties of a color.

Light: <identifier>, <x>, <y>, <z>, <r>, <g>, <b>, {shadows|noshadows}, <contrast>; defines a point light source.

Sphere: <identifier>, <color>, <x>, <y>, <z>, <r>; defines a sphere.

Vertex: <identifier>, <x>, <y>, <z>; defines vertex for planar faces or bicubic patches.

Plane: <identifier>, <color>, <vertex>, ... <vertex> & <vertex>, ... <vertex>; defines edges of a planar face.

Polyhedron: <identifier>; defines a group of planar faces.

Patch: <identifier>, <color>, <r_min>, <vertex_1>, ... <vertex_16>; defines a bicubic patch by 16 control points.

Patch_trees: <identifier>, <color>, <directory>, <file>; reads bicubic patch quadtrees from a file.

Quadric_coeffs: <identifier>, <color>, {actual/auxilliary}, <a0>, ... <a9>; defines coefficients of a quadric surface.

Quadric_bounds: <identifier>, {+|-} <quadric> & .... | {+|-} <quadric> ....; defines bounds of a quadric surface.

Sample: <level>, <diffr>; defines number of pixel sampling levels and division coefficient.

Shade: <maxray>, <hidmin>, <shamin>, <refmax>, <tramax>; sets parameters of the recursive shader.

Status: <what>, <what>, ...; lists current status and contents of

the STRAW data base.

List: <what>, <what>, ...; defines listing options for image generation.

Initialize; initializes the STRAW data base.

Execute; generates a synthetic image.

Exit; exits from the STRAW processor.

Stereo: {left|right}, <x>, <y>, <z>, <angle>; computes complementary stereo camera parameters.

Input: <file>; reads STRAW commands from a file.

Help; lists STRAW command syntax.

Solid_output: <directory>, <file>; opens output solid-parallelepiped file.

Projection_output: <directory>, <file>,{merge_nearrest|merge_closed|merge_orthographic}; opens output parametric-network file and selects the type of projection.