Appendix A

Field functions for point primitives

In the following formulae r denoted a Euclidean distance to a point of interest (x, y, z), i.e., $r^2 = x^2 + y^2 + z^2$. For better clarity, all scaling coefficients that control the width and the height of all function, are set to 1.

A.1. CAUCHY FUNCTION

$$h(r) = 1/(1 + s^2 r^2)^2, \qquad r > 0$$

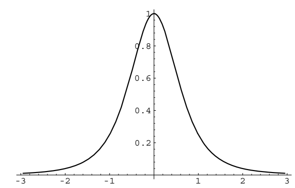


Figure A.1: Cauchy function.

Note: as explained in the text, this is in fact a squared Cauchy distribution.

A.2. GAUSSIAN FUNCTION

$$h(r) = \exp(-a^2 r^2), \qquad r > 0$$

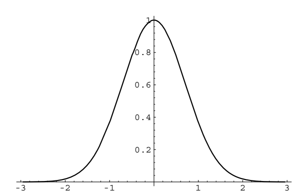


Figure A.2: Gaussian function.

A.3. INVERSE FUNCTION

$$h(r) = 1/r, \qquad r > 0$$

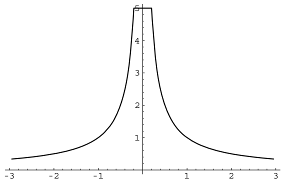


Figure A.3: Inverse function.

A.4. INVERSE SQUARED FUNCTION

$$h(r) = 1/r^2, \qquad r > 0$$

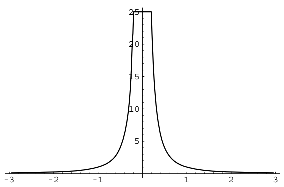


Figure A.4: Inverse squared function.

A.5. METABALLS

$$h(r) = \left[\begin{array}{ll} 1 - 3r^2 & 0 \leq r \leq \frac{1}{3}; \\ \frac{3}{2}(1-r)^2 & \frac{1}{3} < r \leq 1; \\ 0 & r > 1; \end{array} \right.$$

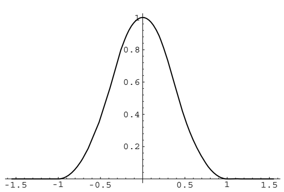


Figure A.5: Metaballs.

A.6. SOFT OBJECTS

$$h(r) = \left[\begin{array}{ll} 1 - (\frac{4}{9})r^6 + (\frac{17}{9})r^4 - (\frac{22}{9})r^2, & r < 1; \\ 0 \quad r > 1; & \end{array} \right.$$

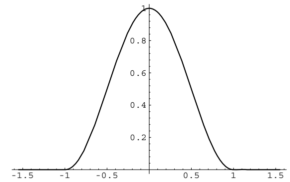


Figure A.6: Soft objects.

A.7. W-QUARTIC POLYNOMIAL

$$h(r) = (1 - r^2)^2, \qquad r < 1$$

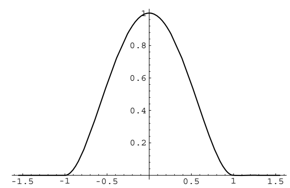


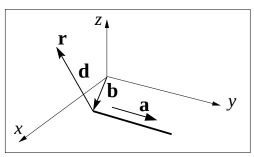
Figure A.7: W-quartic polynomial.

Appendix B

Field functions for line primitives

The following functions describe the scalar fields produced by line segments of length L, convolved with various kernels. The notation used:

$\mathbf{r}$	point of interest (x, y, z)
$\mathbf{b}$	segment base (bx, by, bz)
$\mathbf{a}$	segment axis (ax, ay, az)
$\mathbf{d}$	vector from segment base to a point $\mathbf{r}$
d	length of $\mathbf{d}$
x	dot product of $\mathbf{d}$ and $\mathbf{a}$



Three-dimensional plots show the field distributions in $z = 0$ plane.

B.1. CAUCHY LINE SEGMENT

$$F_{line}(\mathbf{r}) = \\ = \frac{x}{2p^2(p^2+s^2x^2)} + \frac{L-x}{2p^2q^2} + \\ + \frac{1}{2sp^3}(\mathrm{atan}[\frac{sx}{p}] + \mathrm{atan}[\frac{s(L-x)}{p}]),$$

where s defines the kernel width and p and q are distance terms:

$$p^2 = 1 + s^2(d^2 - x^2),$$
$$q^2 = 1 + s^2(d^2 + L^2 - 2Lx)$$

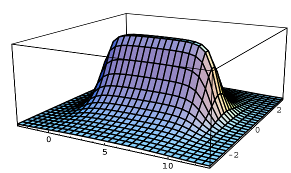


Figure B.1: Cauchy line segment.

B.2. GAUSSIAN LINE SEGMENT

$$F_{line}(\mathbf{r}) = e^{-a^2(d^2-x^2)}(\mathrm{erf}[a(L-x)] + \mathrm{erf}[ax])$$

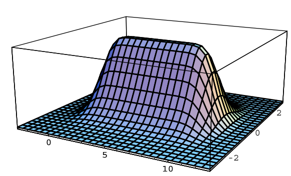


Figure B.2: Gaussian line segment.

B.3. INVERSE POTENTIAL LINE SEGMENT

$$F_{line}(\mathbf{r}) = \ln(L - x + \sqrt{d^2 - 2xL + L^2}) - \ln(d - x)$$

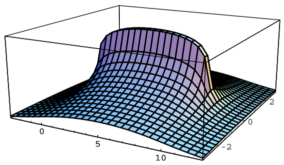


Figure B.3: Inverse potential line segment.

B.4. INVERSE SQUARED LINE SEGMENT

$$F_{line}(\mathbf{r}) = \frac{\operatorname{atan}\frac{x}{\sqrt{d^2-x^2}} + \operatorname{atan}\frac{L-x}{\sqrt{d^2-x^2}}}{\sqrt{d^2 - x^2}}$$

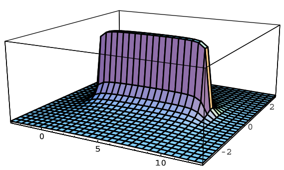


Figure B.4: Inverse squared potential line segment.

Formerly, when one invented a new function, it was to further some practical purpose; today one invents them in order to make incorrect the reasoning of our fathers, and nothing more will ever be accomplished by these inventions.

Jules Henry Poincare (1854-1912)

B.5. POLYNOMIAL LINE SEGMENT

$$\begin{aligned} F_{line}(\mathbf{r}) &= \\ &= R^4((l_2^5 - l_1^5)\frac{1}{5} - \\ &= (l_2^4 - l_1^4)x + \\ &= (l_2^3 - l_1^3)\frac{2}{3}((2x^2 + d^2 - R^2)) + \\ &= (l_2^2 - l_1^2)2x(R^2 - d^2) + \\ &= (l_2 - l_1)(R^2 - d^2)^2), \end{aligned}$$

where R is the width of the kernel and $[l_1, l_2]$ is the integration interval I as shown in Figure 2.8.

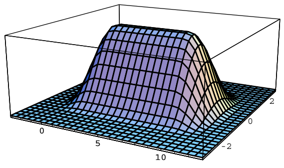


Figure B.5: Polynomial line segment.

Appendix C

RATS Overview and Command Language

C.1. OVERVIEW

NAME

RATS – Ray-Tracing and Animation Tools Software

SYNOPSIS

rats [-/+options] [filename.{art, rat, ice, dat}]

options are:

-?	print this message
-a	use automatic tiling of windows for X displays
-c	use color XPM icons
-d	display traced image line by line
-e	echo on/off
-g	graphics environment on/off
-i	print version information
-j	jitter on/off
-l filename	log all output into a file
-m	memory watcher on/off (may be slow!)
-o file.fmt	set output file name and format
-q	quit after processing input files
-s value	set sampling (number of rays per pixel)
-t	use thumbnail image icons
-v	verbose mode on/off
-w	warning messages on/off
-y	use gray-scale visual for pseudo-color X displays
-r width hight	set horizontal and vertical image resolution

Plus '+' turns the options on, minus '-' turns it off. If no input files are specified, *RATS* read commands from keyboard. Otherwise, *RATS* reads command from input files, that are stored in a LIFO queue; after the last file is processed, *RATS* continues to read commands from keyboard. Command files may be called from inside each other; infinite loop calls are disabled. By convention, input files are expected to have 'art', 'rat', 'ice', 'dat' extensions – they all are treated as batch files.

DESCRIPTION

RATS was born at Caltech in 1994 as a toy ray-tracer written by the author for the *Computer Graphics Laboratory* course CS174. At that time, the course was taught by Jim Kajiya, with a driving force 'of great pitch and moment', which propelled the project far enough to be able to live on its own. Over the years, *RATS*

[1], matured into a complete system for modeling, rendering and animation of conventional and implicit surfaces. Version 7.31 amounts to nearly 60,000 lines of code. *RATS* has been compiled and tested in various flavors of UNIX operating system, e.g. HP-UNIX (Hewlett Packard), Sun OSF1 (Alpha stations), SunOS 4.X (Sun SPARC Stations), ULTRIX (DEC workstations), IRIX (SGI) and Linux (PC).

At a glance, *RATS* has the following features.

- Conventional modeling primitives: arc, box, brick, cone, cylinder, dot, patch, polygon, plane, quadrics, sphere, torus, triangle
- Height fields
- Object hierarchies
- Object instances
- Linear transformations
- Flexible super/under/sampling
- Depth of field
- Motion blur
- Penumbrae
- Transparent shadows
- Solid and flat textures
- Automatic and nested grids
- Implicit modeling primitives: point, line, arc, triangle, plane
- Selective visibility [64]
- Clouds of light and halos
- Surface and texture assignments
- Scalar, color and vector variables
- Internal man pages
- Internal tests for most commands
- Animation tools
- Time profiling tools
- Image arithmetic (AND, XOR, SUB)
- Image viewer/tiler for X displays
- Thumbnail image icons
- Interactive previewer
- TIFF, TARGA, MTV, FLI support

C.2. COMMAND LANGUAGE

RATS command language contains over one hundred commands that are grouped according to their purpose as follows.

RUN	basic commands to start/run/stop the program
CAMERA	create virtual camera
MATERIALS	create textures and surfaces
PRIMITIVES	create primitive objects
OBJECTS	create and manipulate objects
OPTIONS	set rendering/modeling/running/viewing options
SCENE	set lights, backgrounds, atmosphere... and shoot
TOOLS	handy little things: animation, file conversion, etc.

The following naming conventions are used:

`UPPERCASE`	explicit triple x y z or user-defined vector or color variable
`lowercase`	keyword, explicit scalar or user defined scalar variable
`[anything]`	square brackets indicate optional arguments
`Color`	standard color, as LightBlue or user-defined color variable
`fn`	file name, as '../somewhere/somefile.tif'
`sn`	surface name, as 'Plastic'

Many commands have internal test suites and manual pages. Such commands are marked with letters 'T' and 'M', respectively. M-commands usually have complex syntax and some of them require more than one line of comments. If the 'argument' field of an M-command shows dots '...', consult the manual pages for detailed description of the arguments.

[1]The exact meaning of this acronym is still undecided — it evolves with the program and the process is not over. Among the most recent interpretations are Ray-tracing/Animation/Tools/Software, Ray-tracer/Animator/Testbed/System, Rocket/Assisted/Takeoff/System, and Russian/Artists/Can't/Spell.

RUN

Commands of this group control the basic execution of the program.

Opcode	Arguments	Comments	
`read`	`fn1 [fn2 ...]`	read commands from batch file(s)	
`pause`	`[message]`	[print message] and wait for ENTER	
`return`		return from a batch file	
`info`		print version information	
`help`	`[command]`	print help [for a single command]	
`man`	`command`	print the manual page for a command	
`test`	`command [print]`	run a test for a command [print only]	
`log`	`[filename]`	open/close a log file	
`var`	`name = value`	create/update a variable	M
`stopwatch`		start/stop stopwatch	
`quit`		finish the session	

The following characters have special purpose.

Character	Comments
`#`	comment sign
`@`	suppress echo of the following command
`, < >`	delimiters (also space and tab)
{	arguments continue on the next line[s]
}	list of arguments ended

CAMERA

RATS supports two types of camera settings: a conventional type that employs `eyep` and `fov` parameters and more flexible type as described in Foley et. al. [25], via `cp` and `window` command. The other camera-related parameters, such as `aperture, shutter, focus`, etc. are common for both types.

Opcode	Arguments	Comments	
`aperture`	`radius`	camera lens, default 0 (pinhole camera)	T
`focus`	`distance`	to focal plane, default distance to VRP	
`shutter`	`fraction`	of open time, min 0 (no blur) max 1.0	MT
`vrp`	`x y z`	view reference point in world coordinates	
`vup`	`x y z`	view up in world coordinates	
`cp`	`x y z w`	center of projection in VRC	
`vpn`	`x y z`	view plane normal in world coordinates	
`window`	`x X y Y`	min x max X, min y max Y in VRC	
`eyep`	`x y z`	eye position in world coordinates	
`fov`	`hor vert`	field of view in degrees	

MATERIALS

Before any objects are created, their material must be defined. In the text, the `surface` command is referred as `material`, which is more correct conceptually. For compatibility purposes, *RATS* is still using the opcode `surface`.

Opcode	Arguments	Comments	
`texture`	`...`	create/update a texture	MT
`surface`	`...`	create/update a surface (a.k.a. material)	MT

PRIMITIVES

To create a stand-alone primitive, its surface must be specified after the opcode, e.g. `sphere Plastic, 1, 0 0 0`. However, if the primitive belongs to a chain of the same primitive objects (see `chain`), the surface name is omitted.

Opcode	Arguments	Comments	
`arc`	`a b P N a1 a2`	see torus; add start angle and width (deg)	T
`arch`	`b P1 P2 P3`	tube radius + three points P1 P2 P3	T
`box`	`PMIN PMAX`	box defined by min and max points	T
`brick`	`x y z`	box of (xyz) dimensions around the origin	T
`cone`	`b BASE a APEX`	radius at base, radius at apex	T
`cylinder`	`r BASE APEX`	radius, base and apex; lids are closed	T
`dot`	`POS`	polynomial point source	T
`gauss`	`POS`	Gaussian point source	T
`line`	`BASE APEX`	line convolved with a Gaussian potential	T
`patch`	`(V1 N1) x 3`	phong shaded triangular patch	T
`pipe`	`r BASE APEX`	same as cylinder but without lids	T
`polygon`	`V1... Vn`	planar polygon with n vertices (max 128)	T
`plane`	`POS NORM`	plane with a point and normal	T
`quadric`	`...`	create a quadric in local coordinates	MT
`sphere`	`r POS`	radius and center	T
`terra`	`...`	create a terrain out of image file	MT
`torus`	`a b POS NORM`	sweep and tube radii, center, normal	T
`triad`	`V1 V2 V3`	triangle conv. with Newtonian potential	T
`triangle`	`V1 V2 V3`	flat triangle	T

OBJECTS

Primitive objects, created via commands listed above may be organized and manipulated as hierarchies of objects. Objects at every layer of the hierarchy may be accompanied by their transformations.

Opcode	Arguments	Comments	
`chain`	`sn, name`	start a new chain of primitives	T
`object`	`name`	start new compound object	T
`instance`	`...`	clone object	T
`close`		finish current compound object or chain	T
`translate`	`x y z [object]`	along (x y z), last object by default	T
`rotate`	`x y z [object]`	about (x y z)	T
`scale`	`x y z [object]`	along (x y z)	T
`explode`	`x y z [object]`	radially explode an object by (x y z)	
`transform`	`[object]`	force transforms now (to make blur)	

OPTIONS

Options control the output image size and quality, accelerations techniques, memory managements and the way *RATS* interacts with the user.

Opcode	Arguments	**OPTIONS**	**Default**	
`echo`	`on/off`	turn echo on/off	ON	
`warning`	`on/off`	allow warning messages	ON	
`verbose`	`on/off`	run in verbose mode	ON	
`bounds`	`on/off`	use bounding volumes	ON	
`soft`	`on/off`	allow soft objects	ON	
`timer`	`on/off`	do time reports after RT	ON	
`stat`	`on/off`	do statistic report after RT	off	
`map`	`on/off [N]`	create time-profile image	off	
`double`	`on/off`	use double-sided faces	off	
`penumbra`	`on/off`	enable soft shadows	off	T
`noview`	`on/off`	don't display RT image	off	
`nosave`	`on/off`	don't save RT image	off	
`dither`	`on/off`	dither images for viewing	ON	
`gamma`	`value`	gamma correction for viewing	1.0	
`framesize`	`width height`	output resolution	128 128	
`filename`	`name`	output filename	scene	
`format`	`fmt`	save as tif/tga/mtv	tif	
`compress`	`on/off`	compress or not when saving	ON	
`epsilon`	`value`	ray-surface hit precision	1e-4	
`vanish`	`value`	min color value	4e-3	
`maxdepth`	`number`	depth of shading tree	5	
`maxmol`	`number`	max molecule size	300	
`digger`	`[hermite|lagrange|` `brent|ridder|RF]`	who digs roots of isosurface equations?	hermite	
`contrast`	`r g b`	supersample threshold	.25 .2 .4	
`pack`	`...`	set packing method	grid	M
`indicator`	`[none|all|bar|text|` `pixel|line|ETA]`	set RT progress indicator	bar	
`fragile`	`[none|all|arc|pipe|` `cylinder|sphere]`	which soft prims change R as set in the material?	none	
`sample`	`...`	set sampling method	1	MT
`mesh`	`...`	polygonal mesh control	robust	M

SCENE

Objects act on *scene*, which contains lights, atmospheric effects and pretty much the rest of the synthetic world which is built by other commands. For instance, arguments for `report` and `reset` commands are: `data`, `model`, `surfaces`, `textures`, `lights`, `background`, `clouds`, `fog`, `cameras`, `options`, `var`.

Opcode	Arguments	Comments	
`light`	`Color [options]`	create a new light source	MT
`fog`	`Color [options]`	add fog to the scene	MT
`background`	`Color [options]`	create a background layer	MT
`cloud`	`Color [options]`	create a cloud layer	MT
`remove`	`object name`	remove an object from the scene	T
`report`	`...`	display values of most parameters	M
`reset`	`...`	set most parameters to default values	M
`preview`	`...`	visual RT preview for X displays	M
`shoot`	`...`	start ray-tracing of the scene	M

TOOLS

Tools group is a catch-all for utilities that perform various operations on images, control the appearance of the X display, etc.

Opcode	Arguments	Comments	
`animate`	`fn1... [options]`	make animation [loop—mirror—dither]	T
`play`	`fn [options]`	playback animation	
`split`	`fn [frame.fmt]`	split animation into frameNNN.fmt files	
`collage`	`[options] fn1 fn2`	make a collage of image several files	
`fli`	`fn1... [options]`	make a FLI file of [speed N][size W H]	T
`convert`	`fn1 fn2`	convert image file fn1 -¿ fn2	
`resize`	`fn x y`	resize image file fn by x and y	
`diff`	`fn1 fn2 [save]`	SUB two image files and save the result	
`and`	`fn1 fn2 [save]`	AND two image files	
`xor`	`fn1 fn2 [save]`	XOR two image files	
`show`	`...`	display file, color, palette, etc.	M
`kill`	`[w1 w2 ...]`	kill some windows, [all of them]	
`refresh`	`[w1 w2 ...]`	refresh some windows, [all of them]	
`tile`	`[options]`	tile windows on an X display	
`colors`		list available standard colors by names	
`rats`		start a friendly smalltalk	
`table`	`v1 v2 v3 v4 ...`	plot a polyline of up to 256 points	
`shell`	`[options]`	shell dataset generator (make your own)	
`horn`	`[options]`	horn dataset generator	

BUGS and RESTRICTIONS

Certain restrictions are hard-wired into the code. For instance, maximal number of nested input files (128), maximal number of operands per command (512), maximal number of layers per texture (12), maximal depth of object hierarchies (16), etc. When an attempt is made to step over these limits, a 'sorry' message is issued, the command is aborted and the previous state of the program is restored. Several bugs have been spotted but not fixed in time.

AUTHOR

Andrei Sherstyuk

15 September, 1998

SOFT RATS

Version 7.31

C.3. TWO EXAMPLES OF INTERNAL MAN PAGES

C.3.1. Materials

As mentioned before, materials are created using **surface** command, to support compatibility with datasets developed for earlier versions of *RATS*. The following manual pages are copied from the screen as they were produced by the program.

```
RATS> man surface
-------------------------------------------------------------------------------
create a new surface or assigns one surface to another

FORMAT 1:
   surface {                  #    create a brand new surface
       alias                  #    a single word as a surface name (required)
       body        RGB,       #  * main body color
       ambient     RGB,       #  * emitting light
       diffuse     RGB,       #  * diffuse color
       specular    RGB,       #  * highlight spot color
       reflective  RGB,       #  * reflected color
       transparent RGB,       #  * transmitted color
       index       k,         #  . index of refraction, default 1
       dispersion  d,         #  . optical density variation, default 0
       shine       p,         #  . Phong shining
       radius      r,         #  . in isolation for blobby materials, default 0
       blobbiness  b,         #  . ditto
       strength    s,         #  . scale the field, default 1
       skip        RAYTYPE    #    make the surface undetectable for some rays
       texture     t1 [+ t2 + ...], #  add up to 12 layers of textures
   }
FORMAT 2:
   surface new = old        # surface assignment


   -----------
   L E G E N D
   -----------

   RGB may be one of the following:
         1. Color     standard palette color or user-defined color var
         2. RGB       explicit color values (0.25 1 0.75)
         3. number%   percent of body color for diffuse and ambient components
                      percent of incident light for spec, refl and transparent
   RAYTYPE may be one or more of the following keywords: pixel, shadow,
                      reflected, transmitted

         *   fields may be defined via vector variables,
         .   fields may be defined via scalars variables
         all fields are optional, except 'alias'


   ---------------
   E X A M P L E S
   ----------------
      surface Mirror   { diffuse LightBlue, reflective 0.7 0.7 0.7          }
      surface Plastic  { body Red, diffuse 80%, specular 50%, shine 40      }
      surface Wood     { diff LightWood, spec White, shine 100, texture Wood }

-------------------------------------------------------------------------------
Test etude available. Type "test surface" to try it out
```

C.3.2. Textures

```
RATS> man texture
------------------------------------------------------------------------------
create a new texture or assigns one texture to another

FORMAT 1:
   texture {                 # create a brand new texture
       alias,                # a single word as a name for a new texture
       Target,               # required (see below)
       Method,               # required (see below)       default:
       translate  XYZ,       # translate argument         0 0 0
       scale      XYZ,       # scale texture argument     1 1 1
       times      RGB,       # scale texture values       1 1 1
       bounds     RGB RGB,   # clamp texture values       none (-Inf,Inf)
       turbulence t,         # noise in noisy textures    1 (full noise)
       octaves    r,         # level of noisy details    5-6
       mask       1 1 0..    # masking tiles out, if "scale" option was used
                             # there must be X*Y entries, as set in scale
       replace               # keyword: texture values replace target values
                             # (normally scale)
       texture    t1 [+ t2 +...], # add up to 12 layers of other textures
   }
FORMAT 2:
   texture new = old         # create/update a texture using assignment


   -----------
   L E G E N D
   -----------

   "Target" specifies WHAT must be textured. One of the following keywords:
         1. diffuse, ambient, specular, reflective, transparent, index, shine
                    - modify usual  photometric components of the surface
         2. pigment - modify both diffuse and ambient components
         3. normal  - modify normal vector

   "Method" specifies HOW to texture. One of the following keywords:
         1. marble, agate, granite, moon, onion, wood,
            ripples, sandal, checker, paint
                    - use a  predefined solid textures
         2. Color   - apply color values as a texture function
         3. Fu Fv   - use a pair of UV-based functions. Available functions are:
                      one, x, saw, hat, step, sin, cos, sin^2, cos^2
         3. img.fmt - texture values are taken as RGB from the image file. The
                      are two formats, plain and region-dependent:
            img.fmt plain
                    - all points on the surface are textured, using UV values
            img.fmt region <Min Max> <P1 P2 P3 P4>
                    - a point is textured if it belongs to <Min Max> region and
                      its normal vector goes thru the rectangle set by vertices
                      P1 P2 P3 P4

   ---------------
   E X A M P L E S
   ----------------
      texture RedChecker { pigment Red,    scale 2 2 1, mask 1 0 1 0          }
      texture Bumps      { normal  sin cos,  scale 3 4 1                      }
      texture Wood       { diffuse wood, scale 9 8 1,  bounds <DarkWood White> }
      texture MonaTiled  { diffuse monalisa.tif plain, scale 2 2 1, replace    }
      texture MonaBumped   texture   Bumps + MonaTiled

------------------------------------------------------------------------------
Test etude available. Type "test texture" to try it out
```

C.4. SELECTED DATASETS

```
###################################################################
#   cowrie
#
#   Scene       Spindle Cowrie -- a tiny mollusk (at most 4 cm)
#               that lives usually with gorgonians and feed on
#               their polyps.
#   Date        Wed Aug 27 17:29:06 EST 1997
#   Author      Andrei Sherstyuk
#   Features    Blobby arcs and cylinders
#   Comments    Very simple model, produced by the code below
#               The croissant-like combination of arcs turned
#               out to be so good, that inspired the gorgeous
#               model of a coral crab and `Wow' animation.
#
#   Objects     100 cylinders + 3 arcs
#   Version     7.12
#   Time        demiurge, sample 4, 640x480: 24 min 29 sec (v 7.11)
#
#   Nice and simple
###################################################################
@reset    all  # clean up
@echo     off  # no text output, until it's time to trace
soft      on   # enable implicit surfaces
fragile   all  # make sure that material thickness overwrite individual
sample    4    # 4x4 supersampling grid

#
# Camera
#
eyep      3 3 3
fov       50 50
vrp       0 0 0
vup       0 0 1

#
# Output
#
framesize 640 480
filename  cowrie
format    tif

light White point, position 2 4 3, intensity 1.2, noshadow

#
# The radius of spikes-cylinders is declared as
# a variable so I can adjust it interactively
#
var r = 0.05 hot

#
# Blobby surfaces for the body (s1, s2, s3) and the spikes (s0)
#
surface s0 diff White,     spec White, shine 10,  blob -128, radius .034 strength 2
surface s1 diff OrangeRed, spec White, shine 100, blob  -16, radius .125
surface s2 diff DarkRed,   spec White, shine 100, blob   -8, radius .250
surface s3 diff OrangeRed, spec White, shine 100, blob   -2, radius .500

#
# The skeletal model
#
object SHELL
    arc s1 4, 0.125, <-2.5 -2.5 0>, <0 0 1>  5 80
    arc s2 4, 0.250, <-2.5 -2.5 0>, <0 0 1> 15 60
    arc s3 4, 0.500, <-2.5 -2.5 0>, <0 0 1> 30 30
    # read a datafile produced as "cowrie 100 0.75 > spikes.dat"
    read spikes.dat
close
rotate 1 1 0 30, SHELL

#
# Start ray-tracing
#
echo on
```

```
shoot
return

#-------------------------cut here and save as cowrie.c----------------------
/*
############################################################################
#  cowrie.c -- utility to add spikes to the croissant-like body of the shell
#
#  Compile: gcc -o cowrie cowrie.c -lm
#  Use:     cowrie N length > output.dat,
#           where
#           N      - number of spikes
#           length - the length of the spikes
#  Example:
#           cowrie 100 0.75 > spikes.dat
############################################################################
*/
#include "system.h"
#include "types.h"
#include "vectors.h"

/*
 * This is the "body", an arc defined as
 * sweep radius (float),
 * tube radius (float),
 * center (3 vector),
 * normal (3 vector),
 * start angle (float degree),
 * stop angle (float degree)
 * arc s3 4, 0.500, <-2.5 -2.5 0>, <0 0 1> 30 30
 */
#define CENTERx -2.5
#define CENTERy -2.5
#define CENTERz 0

#define START 5.0   /* degrees */
#define THETA 75.0  /* degrees */
#define R     4.0   /* sweep radius */


/*
 * Prints usage and exits
 */
void usage(char *module)
{
    printf("Usage: %s N length\n", module);
    exit(-1);
}

/*
 * Rotate the point about Z axis
 */
void rotate_point(P, a)
Vector          *P;
double           a;
{
    double x, y, sinA, cosA;

    sinA = sin(a);
    cosA = cos(a);

    x = P->x;
    y = P->y;
    P->x = x*cosA - y*sinA;
    P->y = x*sinA + y*cosA;
}




void main(int argc, char *argv[])
{
    Vector   Base, Apex;
    double   len, length, chance, inc;
```

```
    int      N, i;


    if (argc < 3 ||
       ((N = atoi(argv[1])) == 0) ||
       ((length = atof(argv[2])) == 0.0))
        usage(argv[0]);

    printf("# N = %d, length %g\n", N, length);

    for (i = 0; i < N; i++) {
        inc = deg2rad(START + (double)i/(double)N*THETA);
        /*
         * Make the spike as a cylinder "base -> apex":
         * (0,0,0) -> (length, 0,0), then rotate the spike apex aroung Y
         * randomly and then rotate base and new apex around Z incrementally
         */
        chance = 2.0 * PI * drand48();

        /*
         * Apex point is rotated around Y and translated along X
         */
        len = length * sin(2.0 * inc)*sin(2.0 * inc);

        Apex.x = len * cos(chance) + R;
        Apex.y = 0;
        Apex.z = len * sin(chance);

        /*
         * Base translated along X
         */
        Base.x = R;
        Base.y = 0;
        Base.z = 0;
        rotate_point(&Base, inc);
        rotate_point(&Apex, inc);

        /*
         * Ajust the center
         */
        Apex.x += CENTERx, Apex.y += CENTERy, Apex.z += CENTERz;
        Base.x += CENTERx, Base.y += CENTERy, Base.z += CENTERz;

        /*
         * The spike is ready, dump it
         */
        printf("cylinder s0 r, <%g %g %g>, <%g %g %g>\n",
                                          Base.x, Base.y, Base.z,
                                          Apex.x, Apex.y, Apex.z);
    }
}
```

```
#################################################################
# HORSE:    Sea horse for "Modeling Marine Life".
#
# Comments  This model was conceived sitting in the restaurant
#           "Hideout" in Melbourne, where I and Katya went for
#           deserts one night. All the walls were painted with
#           incredibly grotesque seafood samples, including a
#           seahorse that I liked. So here it is.
#           Use: hermite re-used interpolants, precondensed
#           molecules, faster 30/35% than volatile molecules.
# Objects   Soft: 43 arcs
#           Hard:  2 spheres (the eyes)
# Version   7.12
# Precision SINGLE
# TIME:     resulution 320 512, demiurge, sample 4:
#           --------------------------------------
#           hard:   5 min  25 sec
#           soft:  11 min  52 sec
#################################################################
@reset all
@echo off
pack none
fragile all
sample 4

#
# Camera
#
eyep 0 0 20
fov  13 13
vup  0 1 0
vrp  0 0 0

# Output
#
framesize 160 256
filename  horse
format    tif

# Lights
#
light White point, pos -10 5 10 noshadow, inten 0.75
light White point, pos  5 10 10 noshadow, inten 0.25

# Colors
#
var Body  = SummerSky
var Flin  = Gold
var Apple = Gray15
var Ball  = White
var Spec  = White
var phong = 100

var Lo = 0.5 0.5 0.5
var Hi = 0.7 0.7 0.7

texture TT strength sin one bounds Lo Hi scale 22 22 1 times 1.15 1.15 1.15

# Flins
surface F0 diffuse Flin specular Spec, shine phong, blob -256 rad 0.025
surface F1 diffuse Flin specular Spec, shine phong, blob -128 rad 0.10
# Eyes
surface E0 diffuse Ball  specular Spec, shine phong, ambient Gray
surface E1 diffuse Apple specular Spec, shine 50
# Tail
surface T0 diffuse Body, specular Spec, shine phong, blob -64 rad 0.32
surface T1 diffuse Body, specular Spec, shine phong
surface T1 diffuse Body, specular Spec, shine phong
surface T1 diffuse Body, specular Spec, shine phong
surface T1 diffuse Body, specular Spec, shine phong
surface T1 diffuse Body, specular Spec, shine phong
surface T1 diffuse Body, specular Spec, shine phong
surface T1 diffuse Body, specular Spec, shine phong
# Body
surface B0 diffuse Body, specular Spec, shine phong, blob  -8 rad 0.25
```

```
surface B1 diffuse Body, specular Spec, shine phong, blob -16 rad 0.25
surface B2 diffuse Body, specular Spec, shine phong, blob  -8 rad 0.35
surface B3 diffuse Body, specular Spec, shine phong, blob  -8 rad 0.25 text TT
surface B5 diffuse Body, specular Spec, shine phong, blob  -8 rad 0.05 text TT
surface B4 diffuse Body, specular Spec, shine phong, blob  -8 rad 0.25 stren 0.5

object HORSE
object HEAD # nose + forehead + top + back + cheek + eyes (2 spheres) + horns
    arch B0 0.25 { -0.42857 1.8277 0, -0.87857 1.2277 0, -1.37857 0.9777 0 }
    arch B0 0.25 { -0.62857 1.7277 0, -0.37857 2.2277 0, 0.17143  2.4777 0 }
    arch B0 0.25 {  0.18393 2.4777 0,  0.55893 2.4777 0, 0.87143  2.2902 0 }
    arch B1 0.25 {  0.87143 2.2902 0,  1.05893 2.0402 0, 1.12143  1.7277 0 }
    arch B1 0.25 {  0.02143 2.3527 0, -0.1     1.6027 0, -0.86607 1.2277 0 }
    sphere E0 0.15 <-0.50 1.50 0.75 >
    sphere E1 0.10 <-0.50 1.45 0.85 >
    arch F0 0.05 { -0.766 1.615 0.25, -1.016 1.8027 0.25, -1.45357 1.9277 0.25 }
    arch F0 0.05 { -0.766 1.9902 0, -0.95357 2.2402 0, -1.26607 2.3652 0 }
    arch F0 0.05 { -0.641 2.3027 0, -0.82857 2.7402 0, -1.26607 3.1152 0 }
    arch F0 0.05 { -0.328 2.5902 0, -0.32857 2.9277 0, -0.64107 3.3652 0 }
close
object BODY
    # front
    arch B0 0.25 {  1.07143 1.5527 0, 0.72143 1.0027 0, -0.0660701 0.5402 0 }
    arch B0 0.25 { -0.0660701 0.5402 0, -0.44107 0.1027 0, -0.37857 -0.5223 0 }
    arch B0 0.25 { -0.37857 -0.5223 0, 0.12143 -1.0223 0, 0.62143 -1.2723 0 }
    arch B4 0.25 {  0.62143 -1.2723 0, 0.87143 -1.3973 0, 1.12143 -2.0223 0 }
    # back
    arch B0 0.25 { 1.12143 1.7277 0, 0.93393 0.9777 0, 0.62143 0.4777 0 }
    arch B3 0.25 { 0.62143 0.4777 0, 0.49643 -0.0222998 0, 0.87143 -0.5848 0 }
    arch B4 0.25 { 0.62143 0.4777 0, 0.49643 -0.0222998 0, 0.87143 -0.5848 0 }
    arch B0 0.25 { 0.87143 -0.5848 0, 1.24643 -1.1473 0, 1.24643 -2.1473 0 }
    # middle
    arch B0 0.25 { 0.00393 0.1902 0.0, 0.05893 -0.2100 0, 0.63393 -0.9723 0 }
    arch B2 0.33 { 0.52143 0.7777 0.0, 0.06430 -0.0223 0, 0.57143 -0.9848 0 }
close
object TAIL
    arc T0 1.000 0.25 < 0.2   -2.0125 0> <0 0 -1> 0 90
    arc T1 1.000 0.25 < 0.2   -2 0> <0 0 -1>   90 90
    arc T1 0.500 0.25 <-0.3   -2 0> <0 0 -1>  180 90
    arc T1 0.500 0.25 <-0.3   -2 0> <0 0 -1>  270 90
    arc T1 0.250 0.25 <-0.05  -2 0> <0 0 -1>    0 90
    arc T1 0.250 0.25 <-0.05  -2 0> <0 0 -1>   90 90
    arc T1 0.125 0.25 <-0.175 -2 0> <0 0 -1>  180 90
    arc T1 0.125 0.25 <-0.175 -2 0> <0 0 -1>  270 90
    sphere T1 0.25 <-0.050 -2 0>

    arc B5 1.000 0.25 < 0.20   -2 0> <0 0 -1>  290 90
    arc B3 1.000 0.25 < 0.21   -2 0> <0 0 -1>  20 90
    arc B3 1.000 0.25 < 0.20   -2 0> <0 0 -1>  340 60
    arc B3 1.000 0.25 < 0.19   -2 0> <0 0 -1>   30 60
close
object FLIN
    arch F1 0.10  0.93393 1.4777 0, 1.43393 1.7277 0, 1.93393 2.2902 0
    arch F1 0.10  0.93393 0.9777 0, 0.93393 0.7277 0, 1.43393 0.1652 0
    arch F1 0.10  0.93393 1.0402 0, 0.99643 0.8527 0, 1.62143 0.4152 0
    arch F1 0.10  0.93393 1.1027 0, 1.05893 0.9777 0, 1.74643 0.6652 0
    arch F1 0.10  0.93393 1.1652 0, 1.12143 1.1027 0, 1.87143 0.9152 0
    arch F1 0.10  0.93393 1.2277 0, 1.18393 1.2277 0, 1.99643 1.1652 0
    arch F1 0.10  0.93393 1.2902 0, 1.24643 1.3527 0, 2.05893 1.4777 0
    arch F1 0.10  0.93393 1.3527 0, 1.30893 1.4777 0, 2.05893 1.7902 0
    arch F1 0.10  0.93393 1.4152 0, 1.37143 1.6027 0, 2.05893 2.0402 0
close
close HORSE

echo on
shoot
return
```