in

COLLABORATORS			
	TITLE : in		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		April 16, 2022	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

in

1.1 Dust - Dokumentation

```
                ##############################################################################

#                                                                              #
#                Dust V2.42 - Copyright ©1994 by A.Maschke                     #
#                           All rights reserved.                               #
#------------------------------------------------------------------------------#
#                                                                              #
#                       English Documentation V0.5                             #
#                                                                              #
################################################################################
```


(Last changes: 3 November 1995)

1.2 5. Tutorial

Since Version 2.9 Imagine has a new feature called states.

Tutorium 1 - MORPH and Imagine-States
demonstrates
how to create ONE states-object from TWO objects with different point-

and face-count.

If you want to create animations you have to load all the objects created
by Dust into your rendering-software. Because this cannot be done by hand
Dust will help you doing this.
There is a programm called "ISL" creating staging-files for Imagine.
The following

Tutorium 2 - Creation of a 2D-transversal-wave (180 frames)
will demonstrate how to use this program.

Tutorium 3 - Animations with LIGHTWAVE
desribes the creating of scene-files for Lightwave.

The linearcombiation of F/X is demonstrated creating wave-interferences:

Tutorium 4 - Waveinterferences with FOR-loops
.

The last tutorium demonstrates how to program particle-effects:

Tutorium 5 - Programming of Particle-effects: explosions

1.3 Tutorium 4 - Waveinterferences with FOR-loops

To create wave-inferences you have to apply waves to an object sequentially.
Because this cannot be done by hand I implemented the FOR-loops.

Now we want to linearcombine 3 transversal-waves applied to a plane.
180 object will be created.

To do the interference for the first object the followings steps are
necessary:
-load the plane
-create the 1st wave
-create the 2nd wave
-create the 3rd wave
-save the object

We write script doing this:

```
"load(1,plane)
 wave2dframe(1,180,1,2,t,12.0,24,20,30,1.0,60.0)
 wave2dframe(2,180,1,3,t,14.0,36,-30,10,0.9,-30.0)
 wave2dframe(3,180,1,2,t,11.0,22,5,-30,1.1,45.0)
 save(2,hd1:objects/obj.0001)"
```

Now we have to enclose our commands in a loop and to replace the
number 1 with a loop-variable

```
"load(1,plane)
```

```
   for(i,1,180)
    wave2dframe(1,180,i,2,t,12.0,24,20,30,1.0,60.0)
    wave2dframe(2,180,i,3,t,14.0,36,-30,10,0.9,-30.0)
    wave2dframe(3,180,i,2,t,11.0,22,5,-30,1.1,45.0)
    save(2,hd1:objects/obj.%)
   end"

 The variable i runs from 1 to 180, the objects "hd1:objects/obj.0001",
 "hd1:objects/obj.0002",...,"hd1:objects/obj.0180" will be created.

 Entering the command
 "staging3(hd1:objects/obj,1,180,1,180,ram:staging.a)" erzeugt die
 the ISL-staging-file will be created.
```

1.4 Tutorium 5 - Programming of particle-effects: explosions

```
1. Steps
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 1.Scaling
   I think the size of the particles will leave constant so we dont change
   this values
 2.Position
   The position is calculated solving a differential equation, e.g.

    x=f(x0,vx0,eta,t)
    y=f(y0,vy0,eta,t)
    z=g(y0,vz0,eta,g,t)

   (vx0,vy0,vz0:velocity at t=0, x0,y0,z0:position at t=0, g: 9.81 m/s^2,
    eta: Stokes-parameter (usually in the range -0.1...-0.00001.)

   At first we have to calculate the velocity at t=0 for every particle.
   This is done building a vector which starts at the centre of all particle
   and points to the initial particle-position.
   This normed vector is multiplied by the omitted velocity-parameter and
   a random number between 0.8 and 1.2. (This looks more realistic.)

 3.Rotation
   Larger particles should rotate slower than small particles - I think
   this looks more realistic.
   So we have to calculate a boundig-box ore another thing indicating the
   size for every particle.
   (The real size of a particle is (psize.x*scl.x,psize.y*scl.y,psize.z*scl.z))

   But - doing this example we DON'T rotate the particles.

 4.Animation
   The time is our parameter, that means we increase the t-value at every
   frame by a value dt. Then we go with this time-value in our solution
   of the differential equation to calculate the coordinates of the
   particles. (dt=(whole time)/(frames-1))

   (If we want to rotate the particles we have to increase the individual
   rotation-angles by a value da=angle/(frames-1))
```

2. Implementation

- -

All files we should create in this section can be located in the drawer "Tutorial3" of the standard Dust-archieve.
I have supplied source-codes in standard-C (I used Maxon-C), GCC and OBERON.
The script our program should create is named "Tutorial3/PExample.script" so you can compare it with your results.

1. Creating the particle-object

We take a Sphere as structure and a cube a shape-object. After creating the particle-object using Dust we receive the data OCOUNT (number of particles) and PPOS (particle-possitions) to operate.
If you want to change the angles and scaling of the particles you need the data-fields PROT and PSCL, too).
We create a script like this:

```
"load(1,objects/s1)     (load the sphere)
 load(2,objects/c1)     (load the cube)
 o2p(1,2,1,p)           (create the particöe-object)
 savep(1,PExample.obj)  (save the particle-object)
 getocount(1)           (save the number OCOUNT as binary-file)
 !copy T:Dust.output PExample.oCount (rename the file)
 getppos(1)             (save the data-field PPOS as binary-file)
 !copy T:Dust.output PExample.PPOS (rename the file)
 !delete T:Dust.output" (clean up)
```

2. The program

Our program has to load in the binary-files at first. After that it calculates and prints out a Dust-script which will transform and save the particle-objects.
This output is piped into a file which will be executed later.

We create n particle-objects executing the following steps:
- -calculate the particle-positions using the mentioned equations
- -print this particle-positions as SETPPOS-commands
- -print out a save-command
- -next frame

3. Final object-creation

The program name "PExample" is started typing
"PExample >PExample.script".
After that (about 10 seconds) we have to start Dust and to load the particle-object:
"load(1,PExample.obj)".
Now we execute the "calculated" script:
"exec(PExample.script)"
After waiting some seconds we got 12 Imagine-objects.

That’t it !

1.5 Tutorium 3 - Einbindung der Objekte in LIGHTWAVE

```
 Note: I don't own Lightwave - so I couldn't check this out.

 In opposition to Imagine Lightwave loads all objects it recognises
 in the scene-file into your memory. But this shouldn't work creating
 a wave-animation containing of 480 objects...

 I think the following should be good:

 Example: Creating a wave-animation (480 frames)
-------------------------------------------------------------------------

 In the most cases you will build up a big scene containing one
 by Dust transformed object. So the first step is to create this
 scene. After that you have to load one of the Dust-objects,
 e.g. "hd0:tobj.0020", into this scene. The resulting scene-file,
 e.g. "hd1:scene", will be reproduced by Dust 180 times exchanging
 the object-file-names.
 The results are "hd1:scene.0001",...,"hd1:scene.0480".
 Dust will create an additional ARexx-script which causes Lightwave
 rendering the 480 frames, too.

 ATTENTION: The scene must have only one frame (firstframe=lastframe=1)

 The used Dust-command is named "lwstaging". In our example it is
 called like this:

  "lwstaging(hd0:tobj,1,480,1,480,hd1:scene)"

 The Arexx-script "hd1:scene.rexx" is executed typing

  "rx hd1:scene.rexx".

 Notes: 1. After changing your scene you have to rerun the
           "lwstaging"-command.
        2. object-filenames are case-sensitive !
        3. You can use the objects in the order "hd0:tobj.0480",...,
           "hd0:tobj.0001", too.
        4. Suggestions are welcome.
```

1.6 Tutorium 2 - Creation of a 2D-transversal-wave (180 frames)

```
-create a plane (20*20 points), rotate it along the x-axis (90 degrees)

-load this object into Dust, create 180 wave-objects typing:
  "wave2d(1,180,hd2:obj,t)"

-enter Imagine, create a new project called "wave",

-create a perfect scene using the stage-editor, save it

-load one of the Dust-objects into your scene, e.g. "obj.0001",
 adjust the camera, add lights,...
```

```
-enter the action-editor, set the "highest frame"-number to 180,
 scale the time-lines of all actors except "obj.0001"

-save it and copy the resulting staging-file "wave1.imp/staging"
 into the T:-drawer

-call the "ISL"-program "destage" typing
  "destage t:staging  t:staging.a"

-load the resulting ascii-file "t:staging.a" into an editor

-Search for a line looking like this (this is a ISL2.0-file, some
 things will be have changed):
 "ACTOR FRAMES 1 180 NAME "hd2:obj.0001" CYCLE 0. 0. TRANSITION 0"

 The trick is to replace this line by the object-sequence:

-enter Dust and type
  "staging3(hd2:obj,1,180,1,180,ram:tt)".

-insert the file "ram:tt" after killing the line mentioned above, now
 you should got 180 lines looking like:

  "ACTOR FRAMES 1 1 NAME "hd2:obj.0001" CYCLE 0. 0. TRANSITION 0"
  "ACTOR FRAMES 2 2 NAME "hd2:obj.0002" CYCLE 0. 0. TRANSITION 0"
  "ACTOR FRAMES 3 3 NAME "hd2:obj.0003" CYCLE 0. 0. TRANSITION 0"
                                    .
                                    .
                                    .
  "ACTOR FRAMES 180 180 NAME "hd2:obj.0180" CYCLE 0. 0. TRANSITION 0"

-save it

-re-convert it into the Imagine-format using "restage"
  "restage t:staging.a  t:staging".

-copy the file into your project-drawer:
  "copy t:staging hd1:Imagine/wave1.imp/staging"

-go back into the stage-editor, render a preview, render it...

NOTE: We could write a shell-script which performs this steps, too.

   .key project,obj,from,to
   .bra {
   .ket }
   ;this script creates an Imagine3.1-staging file for an object-sequence

   ;Parameters:
   ; project: full path of the Imagine-project, e.g. "hd1:imagine/wave.imp"
   ;     obj: object-base-name with path, e.g. "hd0:obj"
   ;    from: first object, e.g. 1
   ;    last: last object

   ;Example: stageit hd1:Grafix/Imagine3.1/test.imp shit:obj 1 12
```

```
;Required software: Dust, ISL3.x, CEd and Ed (quickstarter) in your C:-drawer,
;the Rexx-Master must be active, too.

;init
clear
set echo=off

echo "Creating a backup of the old staging-file"
copy {project}/staging {project}/staging.o

echo "Creating a-Dust-script"
echo >T:Dust.tmp "staging3({obj},{from},{to},{from},{to},T:staging.tmp)"

echo "Running Dust"
Dust T:Dust.tmp

echo "Running Destage"
destage {project}/staging T:staging.a

echo "Running CEd"
 Ed T:staging.a
 rx "address 'rexx_ced' 'search for {obj}.'"
 rx "address 'rexx_ced' 'Beg of line'"
 rx "address 'rexx_ced' 'Delete line'"
 rx "address 'rexx_ced' Include file 'T:staging.tmp'"
 rx "address 'rexx_ced' 'Save'"
 rx "address 'rexx_ced' 'Quit'"

echo "Running Restage"
restage T:staging.a {project}/staging

;clean up
delete >NIL: T:Dust.tmp T:staging.a T:staging.tmp

echo "Bye !"
```

1.7 4. DOCUMENTATION

1.8 4.5 The ARexx-port

```
You want to control Dust via other applications or GUIs ?
So you need an ARexx-Port.

    NAME: "Dust"
COMMANDS: PARSE <cmd> - let Dust parse the string <cmd>,
                        all Dust-commands except "EXIT" are allowed
          EXIT        - leave the ARexx-mode

To enter the Dust-ARexx-Mode simply type "REXX".



ARexx-Examples via "rx":

 rx "address 'Dust' 'EXIT'"
 rx "address 'Dust' 'PARSE ?'" (invokes general help)
 rx "address 'Dust' 'PARSE load(1,s1)'" (loads an object)


Other example:
You wrote an GUI and want to start and quit Dust in the background.
So write a batchfile "rexx.dust" containing the lines:

 rexx
 exit

and start Dust typing "Dust rexx.bat".
So Dust will execute your PARSE-calls until you call the ARexx-EXIT-
command. After that the batch-execution will be continued = exit.

If you don't want to get any text-output from Dust you have to
set the QUIET-parameter.
```

1.9 4.3. The Preview

```
You can open as many windows as you want showing one object. (You can
have as many objects as you want, too.)
Every window is one process-so the preview gives you an impression
of real multitasking. If you kill an object all windows drawing this
object will close automatically, if you change an object all windows
will redraw. You can change the attributes (drawmode, size, position)
interactively pressing a shortcut or using Dust-commands.
(See the example "Windows.bat")
So every window has an identifier.

You can open the windows on the default every public-screen or on
an own Dust-Screen, upto 256 colors can be used.

There are the following drawmodes:

 - Bounding Box : fast rotating, zooming              (<b>-key)
 - Wireframe                                          (<w>-key)
```

```
 - Solid                                                    (<s>-key)
 - Color        : uses face-colors, 1 lightsource           (<c>-key)

The mode Color requires OS3.0.

Every windows has some additional shortcuts:

 - cursor-keys       : rotate
 - <Del>/<Help>-key  : zoom
 - <Esc>-key         : close
 - <o>-key           : toggle outlined-flag
 - <a>-key           : show recent view-angles, zoom, ...
 - <p>-key           : toggles perpective
 - <r>-key           : rescale the window (if KEEPSCALE=true)

There is no preview for particle-objects - you must convert them
in 3D-objects using P2O.
```

1.10 4.2 Concept

```
                The program can hold as many objects in buffers as you want.  ↩
                   Every
object has a number which is called objectID. All commands manipulating
objects require such an objectID as argument.
All structures in Dust get an
               ID
                (identifier) which is
a number greater or equal zero.

Dust is a kind of a programming language so it interpretes
commands, knows mathematical expressions, global variables
and can perform loops.
All commands have an unique structure:
 identifier([<argument1>[,<argument2>[,...]]]).
Identifiers are lower case, arguments enclosed in "[" and "]" are
optional.
Every argument has to be of a Dust-related
               type
               .

All numeric arguments can be omitted as mathematical expressions
containing global variables, rounding is done automatically, e.g.
 "load(4/3,obj)" is identical to "load(1,obj)".

You can define global variables using the CALC-command or the
online-calculator-mode, e.g.:
 "calc("a=3*sin(46)")" or
  ".a=3*sin(46)"

Strings, e.g. filenames can contain special
               format-characters
               .
If a Dust-command recognises such a format-character it inserts the
recent value of the
               loop
```

```
               -counter.
The loop-counter is equal to the loop-variable of the innerst FOR-loop or
the value ACTVAL outside of loops.
Example:
 "load(1,obj.%)" will load "obj.0034" if the loop-counter is 34.

If you want to lineracombine some F/X you will need loops and the
FRAME-commands.
Example:
 The command
  "XYZ(<objectID>,<n>,<filename>)"
 performs such an effect creating n objects. In case of a linear-
 combination it's not necessary to save all object to disk.
 Usually there is a complementary command of the structure
  "XYZFRAME(<objectID>,<n>,<i>,<dest>)" which only creates the
 i-th object of the sequence and stores it to buffer dest.
For more explanation the example-script "interfere.bat".

Objects can be saved and loaded in several formats, you can
set your favourite saver-module with the
               SET
               -command.

If you want to write extended batch-files you will need an

               IF
               -construct.

There is a detailed
               explanation of the features
               .
```

1.11 Object-formats

```
The supported file-formats are the following:

  1. TDDD - this is the standard-format read and written by Dust

      Features:
       -all Attributes can be read, written, modified
       -Imagine3.0-Textures and -Brushes can be read, written, modified
       -every face has its own color (CLST)
       -hard/soft edges will be treated correctly
       -subgroups are supported
      Restrictions:
       -reflection-maps are skipped (BRS4 ?)
       -the reflection- and transluceny-parameters are taken from
        the global atttributes (means: no RLST- and TLST-handling)

  2. LW (Lightwave)

      Features:
       -polygons are assumed to be convex and converted into triangles
       -surfaces are converted into Imagine-Subgroups
```

```
     -the DOUBLESIDED-flag can be set by the BACKFACES-parameter
     -the specular level will be converted
     -storing all textures/brushes/...
    Restrictions:
     -no lines (polygons with 2 points) and point-polygons

3. MC4D (MaxonCinema4D)

    Features:
     -triangles, conversation of quadrangles into triangles
     -converting of axis-attributes
     -chunks with the identifier 8 are read and written
    Restrictions:
     -materials aren't supported because of no documentation
     -chunks of type 5,6,7 are skipped because of no documentation

4. VS (Videoscape) - loading and saving is slow (ASCII-format)

    Features:
     -Videoscape-colors are calculated as good as possible
     -creation of backfaces if the BACKFACES-parameter is set

5. SPHERES - TDDD-Group-Objects made of real Spheres
    This is a special particle-object-format (only for Imagine-users)
    and can only be written.

6. PARTICLE - The fast DUST-Particle-Format
    This Format consists of the particle- and the object-information
    in TDDD-format. So all attributes are supported.

The LOAD-command automatically recognises the format, all object-saving
procedures (e.g. SAVE or MORPH) create objects in the format specified
by the SFORMAT-parameter.

NOTE:  Dust converts LW-Surfaces into Subgroups and Subgroups into
       LW-Surfaces - most of the object-converters (e.g. Vertex, Pixpro,
       Castillian) aren't able to do that.
```

1.12 Data-types

```
              Dust knows the following data-types:


             Identifier:
              integer greater or equal than zero.

     Real:
             mathematical expression
              containg global Variables,
           functions, ...

  Integer: mathematical expression, rounding is done automatically

   String: string containing format-commands.
```

```
    Filename: String oder "", in case of "" a filerequester is opened
```

1.13 Mathematical expressions

```
 All Dust-commands allow mathematical expressions instead of
 plain numbers, other commands require expressions containing
 special variables:

  1. Special parameters required by the FUNC-commands:
    X0 - initial x-coordinate
    Y0 - initial y-coordinate
    Z0 - initial z-coordinate
    T0 - parameter
  Example: You want to plot the 2D-Function sin(x^2+y^2):
   -load a plane (dimensions -50..-50,-50..50,0..0)
   -now enter "func(2,"30*sin(x0*x0/30+y0*y0/30)",0,z)"
  This uses the x- and y-coordinates to modify the"
  z-coordinate of the plane."

  2.Constants
    'pi', 'ee' (2.71..)

  3.Operators
    '+','-', ':-(', '/', '^'

  4.Functions
     'jump','entier','int','abs','sqr','sqrt',
     'exp','ln','log','log10','log2','tentox','twotox',
     'sin','arcsin','cos','arccos','tan','arctan',
     'sinh','cosh','tanh','artanh', 'degtorad','radtodeg',
     'rnd','fac','ceil','floor','round');

 NOTES: 1. All angles in degrees.
```

1.14 Identifier

```
 Dust knows the following identifiers:

    objectID : objects

  particleID : particle-objects

    windowID : opened windows

     brushID : brush of an object

   textureID : texture of an object
```

1.15 Loops

```
                Dust can perform nested FOR-loops, the loop-counter is equal to  ↩
                   the
loop-variable of the innerst loop.
The loop-counter is the number inserted into a string containing

               format-commands
               . Since version 2.3 you have
the choice to place a numeric argument enclosed in brackets after the
format-command.

Syntax:

 FOR(<loop-variable>,<from>,<to>[,<step>])
  .
  .
  .
 END

Example:

 for(i,1,10)
  echo("unformatted:$, formatted:%")
 end

or

 for(i,1,10)
  echo("unformatted:$(3*i), formatted:%(i+1)")
 end
```

1.16 Format-Commands

```
Contains a string format-commands so this commands will be replaced by
the recent value of the loop-counter or the specified numeric argument.

 $ - inserts the value unformatted
 % - inserts the value formatted

Example:
1. A string "$th object: obj.%" is changed into "23th object: obj.0023" if
   the loop-counter was 23.
2. A string "$(3*i+1)th object: obj.%(12*i)" is changed into "7th Object: obj ↩
    .0024"
   if the global variable i was set to 2.
```

1.17 4.1. Installation

```
At first you have to copy the keyfile "Dust.key" and the config-file
called ".dustrc" into you S:-drawer.
```

```
Dust needs a drawer containing the online-help-texts. By default this
drawer is searched as "DustHelp" in the work-directory.

But if you copy Dust into your logical c:-drawer and want ot start it
from everywhere you have to specify a complete path.
E.g., your help-drawer is located as "HELP:Dust" you have to change
the config as follows:
 1. Start Dust
 2. "set(helpdir,help:dust)"
    "saveconfig"
    "exit"
 3. Restart Dust and enter "?"
Now you should see the general help-page.


Notes:   1. Dust needs the following libraries

                asl.library oder arp.library
                mathieeedoubbas.library
                mathieeedoubtrans.library
                mathtrans.library
                xpkmaster.library
                xpkIDEA.library
                rexxsyslib.library

            and the following DOS-commands in you C:-drawer

                delete
                rename
                execute.

         2. You should run Dust from a shell not from Workbench,
            the stack-size has to be about 30000 bytes.

         3. It's recommended to use the following tools to make life easier

                KingCON (history, scroll-bar, filename-completion),
                Powersnap (cut examples from the online-help)
                XSize (window-sizing like UNIX)
```

1.18 1. Program-description

```
                Dust is a F/X-software manipulating 3D-objects. So it's a perfect
completion for your modelling-program and comes with many features
no modelling-program has.

Because Dust is no toy the user should know how to use a 3D-rendering
and -modelling-program and should have basic programming-skills.

Because there are lots of good 3D-programms for the Amiga I had to
made a choice - Dust supports only "Imagine" and "Lightwave" directly.
If you use another program you have to obtain a good object-converter
which is able to operate on batch- or ARexx-scripts.
(You cannot convert 120 objects by hand...)
```

```
The main features of Dust are:

 -local metamorphosis which produces very good results

 -smooth-algorithm producing incredible results

 -direct support of "Imagine" und "Lightwave" - the most popular
  raytracing-packages

 -loading, saving and viewing of object-sequences

 -metamorphosis between any two objects

 -particle-system: exetremely simple structure (for programmers),
                   opens a new dimension of object-modelling

 -realistic explosions (gravity, stokes-friction, ...),

 -realistic water-waves (3D-unharmonic waves)

 -1D-,2D- and 3D-waves, transversal, longitudinal, including particle-waves

 -mathematical distortion of points, face-colors, particles,
  so you can use Dust as simple function-plotter, too

 -every programmer can create various particle-effect in an easy way,
  the knowledge of the TDDD-format (or object-structures etc.) is not
  necessary

 -every face can have an own color, every edge has a sharp/soft-flag,
  this data is kept creating particle-objects

 -handling and changing of Imagine3.0-textures and -brushes

 -sphere-objects: particle-Objects made of real spheres

 -creation of objects from external programs

 -ARexx-port

 -user-defined variables, mathematical expressions instead of
  plain numbers as arguments, loops, simple if-statement

 -very flexible preview-function

 -online-help

 -command-, parameter- and help-topic-completion

Other programs, e.g. Imagine which is supported by Dust directly, are
able to create some of effects mentioned above, too. But this programs
create the effect while rendering - you cannot edit the transformed
objects. With Dust you can linearcombine all types of effects.

There is a more detailed
               explanation of the features
               .
```

1.19 2. System- and user-requirements

Dust is made for the semiprofessional or professional user of 3D-programs which has reached the borders of the usual software. Knowledge of a programming language is strongly recommended (ARexx should be enough).

Professionals will recognize the flexibility of Dust - real beginners will have no fun.

Because Dust is made to be used together with "Imagine" or "Lightwave" a power-Amiga is required. (Dust should work on any Amiga - but there is no reason to run it on a A500).

A serious user should have the folling configuration: 10 MB RAM, 100 MB free and fast HD-partition, AGA, OS3.1

NOTE: Dust is a program to play with, too. You CAN morph a sphere into a torus on an A500 and watch at the results using the preview-function of Dust. But you CAN'T render an animation using Lightwave.

1.20 3. Program-status and licence

Dust is SHAREWARE with a fee of $25 or 25 DM ("Deutschmarks").
Registered users will receive a keyfile which contains some data Dust needs to work and some encrypted private data of the user.
All Dust-related files except the registered keyfile can be copied and spread freely.
The first user (the registered user) of a spread keyfile can be indentified easily decrypting the keyfile-data...

Registered users will get the warranty, that I try to remove detected program-bugs as fast as possible. There are no other warranties I am able to give you.
Because I don't ship disks you will receive updates or keyfiles through the Aminet or via EMail.

All use of Dust is made at own risk !

Important notes

The unregistered version (the version you can download from Aminet) has some commands disabled. The list of the affected commands is supplied in the file "README2".

Owners of older versions of Dust receiving an update should read at first the files "README" and "HISTORY" because there isn't a real features-list

in this documentation.

1.21 7. Address of the author

Andreas Maschke
Zenkerstraße 5
06108 Halle/Saale
Germany

Phone: ++49 (0)345/5170331
EMail: epgbc@cluster1.urz.Uni-Halle.DE

1.22 6. Copyrights

1.23 Explanation of the features

1. Metamorphosis between any two objects (MORPH,PMORPHx,MORPH2,CREATEFACES, ↩
   SORTFACESx,SORTPOINTSx,MORPHSGROUP)

   The following four methods I developed:

   -Local Deform-Morphing: (CDEFORM, CDEFORMINTERP)
   Local Morphing means morphing of curves (parts) of the
   source-object into another curves of the destination-object.
   The destination-curve can be interpolated using two real
   curves of the destination-object.
   Because only the source-object is deformed you can later use

```
    the morph-feature of your renderer to create the metamorphosis.
    (e.g. the STATES-function of Imagine)

    The idea behind this comes from the 2D-morph-programs
    like "MorphPlus". Here you have to specify which portion of
    the source-Image is to morph into which region of the
    destination-Image. Only this method can produce good results
    because the "stupid" computer doesn't know what he is doing.

    There are some additional commands and parameters which help
    you using this commands.


    Because only the source-object is deformed you can later use
    the morph-feature of your renderer to create the metamorphosis.

    The idea behind this comes from the 2D-morph-programs
    like "MorphPlus". Here you have to specify which portion of
    the source-Image is to morph into which region of the
    destination-Image. Only this method can produce good results
    because the "stupid" computer doesn't know what he is doing.

    There are some additional commands and parameters which help
    you using this commands.

  -Global Deform-Morphing: (DEFORMMORPH)
    This is a very strong and slow algorithm which often produces
    very good results. Because it performs some kind of iteration
    the number of created objects is unknown - but you can specify
    the minimum. The final number of objects is stored to the
    variable "result".

    For good results the differences between the objects should be small.
    (If you want to morph a tricycle into 25 balls you have to use the
    Triangle-Morph-algorithm.)
    Additional note: In the most cases this procedure won't produce
                     usable results. Therefore I developed the
                     local deform-operator which always produces
                     very good results - but it needs YOUR help.

  -Triangle-Morphing:
    procedure PMORPH
     -creates two objects of the same triangle-count
     -searches the closests triangles and resorts them
     -searches the closest points and resorts them
    procedure MORPH
     -executes PMORPH
     -executes a linear Morph (Imagine can do it for you,too)

  -Build-Morphing
    -kill the Source-Object while building the
     Destination-Object:
      linear: BUILDMORPH
      randomly: BUILDMORPHRND

OBJECTIVE RATINGS:
 CDEFORM:     very good, no other algorithm can produce better results
```

```
 DEFORMMORPH: works only sometimes, results may be good
 PMORPH:      looks very bad, works always
 BUILDMORPH:  may produce "beam-me-up"-effects, very seldom used
```

2. Object-Smoothing (SMOOTH)

   This algorithm interpolates the surface od an object using splines
   along the edges - you can increase the quality of your objects
   by doing nothing

3. Explosions (EXPLODE)

   Dust creates more realistic explosions than Imagine does:

    -small particles fly wider than larger particles
    -small particles rotate faster
    -all particles are flying up to z=0 (ground) because of the gravity

4. Waves (WAVE1D,WAVE2D,WAVE3D)

   Many types of harmonic waves can be applied to objects:

    -1D, 2D and 3D (spherical or faked)
    -transversal or longitudinal

   The following parameters can be adjusted

    -wavelength
    -amplitude
    -wave-center (source)
    -damping
    -phase

   There are wave-procedures which are guessing the "best" parameters.

 4.1. Interferences

    You can create wave-interfences (water) linearcombining some
    waves, see Tutorium 2

5. Particle-system (P2O,O2P,O2S)

   Dust creates particle-objects from two ordinary objects. The particle-
   positions, -angles and -scaling can be written as binary-file.
   So every programmer can create his own particle-effect, only the
   positions,... must be modified - the final object-creation is done by
   Dust - see Tutorium 3.
   Implemented effects are waves, explosions and mathematical distortions.
   (Erzeugen eines "echten" 3D-Objekts usw.) übernimmt Dust.

6. Mathematical distortions (*FUNC,*CFUNC,*P*FUNC)

   Points, face-colors and particles can be modified using mathematical
   formulas. The following parameters can be used in the expressions:
    -initial coordinate (x0,y0,z0)
    -t0 (animation-parameter).
   For every object-dimension you can specify one formula - so you will never

reach limits...

Example: You want to plot the function "sin(x^2+y^2)". So load a plane
into Dust and type "func(2,30*sin(x0*x0/30+y0*y0/30)",0,z)".
(the scale-factors depend on the plane-size)

7. Gravity (PFALL,PFALL2)

All points of an object fall down upto z=0.
There are the following ways to do this

-for every point the same force (gravity-force)
-for any point a individual force depending from the distance
between the point and z=0.

8. Realistic water-waves (WATER,WATERFRAME,WATERZ,WATERZFRAME)

One or more rings are growing from the centre, this kind of wave
is unharmonic and 3D. The parameters

-amplitude
-wavelength
-wave-source
-damping
-number of rings
-number of times wave-growing is performed (=maxsize of the rings)

can be specified - there is a "best-guess"-function, too.

The WATERZ-procedure moves the points only in z-direction - this
isn't as realistic as WATER - but looks pretty good, too.
(Why ? Sometimes the 3D-motion causes errors if the amplitude
was too large.)

9. Misc

LATTICE creates a lattice-like surface

ADDFACE creates objects adding faces

RENAME renames object-sequences, e.g. "ram:obj.0001 .. ram:obj.0023"
into "hd1:obj.0023 ... hd1:obj.0001".

STAGING2/STAGING3 creates ISL2.0/ISL3.x-staging-files to make
Imagine-staging easier - see Tutorium 1

LWSTAGING creates multiply scene-files from one exchanging object-filenames,
an ARexx-render-script is created,too

TXTDIR/BRSDIR changes directory-names of all textures/brushes of an object

BUILD(RND) kill sequentially/randomly faces upto one

DISTORT moves randomly points of an object

TRIANGULATE creates points and edges for every face

```
    MERGE merges

    SCALEFACES scales faces along their centre

    ...
```

1.24 4.4. The particle-system

```
                The particle-system is one of the strongest features. Every ↩
                   programmer can
 do his own effects using the simple data-structures created by Dust from
 external programs. The external program only has to calculate the
 particle-positions, -angles and -sizes - the 3D-object is created using
 Dust.

 There are some effect implemented, yet.

 To create a particl-object normally two objects are required:
  -the structure-object: defining the particle-positions etc. and
  -the shape-object: defining the particles

 There are two methods to calculate the positions, angles and size
 of the particles:
  -the face-method(FACE): every face of the structure-object "gets"
   one particle located in the face-centre
  -the point-method(POINT): every points "gets" one particle

 The alignment is be done rotating the particle that its x-axis and
 the face-normal are parallel. If you don't want this alignment you
 have to set the ALIGNP-option.

 Dust creates a
               data-structure
                containg the positions, ... and the
 particle. This structure can be loaded and saved in a very efficient
 way and is called particle-object. To render such an particle-object you have
 to cenvert it into an ordinary object using the P2O-command.

 Particle-objects can be copied, loaded, killed, ... like 3D-object,
 but rotating, scaling,... affects only the particles.

SPHERE-OBJECTS

 Sphere-objects are a special kind of particle-objects. Their
 shape-object is simply a "real" sphere. Because you cannot convert
 this type of object into polygons it can be only saved as
 Imagine-group-object which Dust cannot load. There is the
 parameter SAVESPHEREP which causes Dust to write an additional
 particle-file with the suffix ".dpo" while saving such a group-object.

 While creating a sphere-object using O2S the attributes of the
 structure-object are applied to the spheres - so textures etc.
 are possible.
```

1.25 The Dust-Particle-format

Dust gives every programmer the possibiliy to create particle-effect in
a very efficient way.
The 3D-objects are created in Dust and manipulated from the external-
program writing batch-files or using the ARexx-Port.
So the external program must read binary files, calculate some data
and write some text - I think this should be possible using every
programming language. A tutorial in C, GCC and OBERON is supplied.

A particle-object contains
 -the positions (PPOS)
 -the rotation-angles (in degrees) (PROT) and
 -the scaling (PSCL) for every particle.

To create an effect you have at first to read in some of this data-fields
to get initial values. Using this values you calculate the new and
print out it as Dust-commands. That's it.
The most important value is the number of particles - called OCOUNT.

There are the following commands to write binary-files
 GETPPOS, GETPPROT, GETPSCL, GOCOUNT and GETPSIZE.
To modify the particles you use some of the commands
 SETPPOS, SETPROT and SETPSCL.

Here are the steps to create a full effect:

 1. create a Dust-script which creates the particle-object and
    outputs the data-fields (PPOS,...)
 2. external program:
     -reads in the binary data
     -calculates the effect and prints out the new values formatted
      as Dust-commands,
      the output is piped into a file
 3. the resulting script is processed by Dust.

1.26 4.6 COMMANDS

It makes no sense to include here the command-description you can
find in the online-help. (No man can update such a lots of files.)
Enter Dust and type

 "?commands"

to get a list of all commands.

To get information about a command type:

 "?<commandname>", e.g. "?load".

1.27 4.7 PREFS

```
It makes no sense to include here the prefs-description you can
find in the online-help. (Nobody can update such a lots of files.)
Enter Dust and type

 "?prefs"

to get a list of all settings.

To get information about a parameter type:

 "?<parametername>", e.g. "?zoom" .
```

1.28 Tutorium 1 - MORPH and Imagine-States

```
We create a states-object containing two states: a ball and a cube.
At first we have to preprocess this objects using Dust's PMORPH-function:
 load(1,c1)
 load(2,s1)
 pmorph(1,2)
 save(1,m1)
 save(2,m2)
After that we can start Imagine and load the objcts "m1" and "m2" into its
Detail-editor.
After selecting the ball we choose the function States/States/Create
and change the name "DEFAULT" into "BALL". In the data-type-requster
all options should be selected.
Now we add the state of the cube to the sphere-object selecting
States/States/Import and clicking at "PLANE" (or the name of your cube-object)
in the Objects-window. The name of the state should be "CUBE".
(Don't forget to select the data-types in the data-types-window - usually
 I do select all.)
Now the sphere-object can be saved.

To test the states select the sphere, choose the menu States/States/Tween
and select "CUBE" in the states window.

NOTE: You can create more impressive states-objects using the CDEFORM-
      command - that's clear.
```

1.29 The IF-Construct

```
I tried to make the implementation as easy as possible because
I didn't want to change the (very complicated) Dust-kernel.

     Usage: IF(<expression>,<command>[,<alternative command>])
  Examples: if(1<2,echo(true),echo(false))
            if(xmin>t,.xmin=t)
            if(a,echo(a isnt 0),echo(a is 0))
     Notes: 1. Only simple expressions containing the boolean operators
```

```
    "<=", ">=", "<", ">", "=" are allowed ("==" is the same as "=")
2. Writing in C you would say:
    if(<expression>) <command>; else <alternative command>;
```