DS_EXPT - An implementation of a direct search algorithm for the computer design of holograms

Matt Clark

March 4, 2005

1 Release notes

This is ds_expt (c) Matt Clark September 2002 This release is covered by GPL. The license statement is located at the end of this document in section 8 and full details were included in this distribution in a file called COPYING.

1.1 Revision history

1991-1994 Many versions
1995 ds
1996 ds_bin
1997 ds_ap
1997 ds_new
1998 ds_gs
1998 ds_expt version 0.9
2002 ds_expt pared version 1.0 GPL public release.
2005 ds_expt pared version 1.1 GPL public release (fixed to compile of modern compilers).

2 Introduction

2.1 Where it comes from

This program is a synthetic hologram or computer designed hologram design program. It implements an algorithm described in Applied Optics [1]. This algorithm is in turn based on the state-variables direct search method described in Optics Communications [2]. If you publish work based on this algorithm please cite these two papers rather than the program. The program can implement either algorithm. It implements the state-variables and grey-scale modified state-variables costs functions, fringe follow / pixel preselection, memory efficient random exhaustive searching and optimal stopping and starting functions with N+1 start states and N finish states with mopping up at the end of sweep 1 and probabilistic run length optimisation.

Many other things have been implemented in the past but this is all you really need.

2.2 What it does and doesn't do

This program implements the modified state variables direct search algorithm for the computer design of holograms.

If you are new to this you will need to get your head around some concepts before starting, just in case,

This will not allow you to; 1) Make holograms - you need a fabrication facility. 2) Make / design pictorial holograms or display holograms or anything resembling a nice picture you can actually look at (in principle you can but no design or fabrication facility on earth would permit you to make anything worthwhile here - I (matt) have looked into my (rather small) hologram apertures and directly observed the image in space - I don't recommend this unless you are very confident about your exposure / eye safety levels). 3) Produce a 3d display on you computer.

What it can do is to produce a highly optimised quantised (fabricatable) phase distribution tailored to your optical input which produces a near optimal image. The image can be three dimensional and it can contain many levels of intensity (grey scaling). If you fabricate the hologram the real world performance will arbitrarily closely match the design performance (or the simulation performance) (there is no loss of performance between design and fabrication provided you fabricate accurately).

2.2.1 Properties of the direct search

- Designs are stand alone- no additional optics are required to reconstruct them - they focus themselves.
- The designs achieve diffraction limited optical performance - the diffraction limit is defined by the hologram aperture and the working distance.
- The diffraction efficiency is close to the theoretical maximum for the type of hologram designed. By "close" I mean that it is close enough for the precise definition of efficiency to matter more than any measurable loss of efficiency.

  (if you are very unlucky and uncover a special symmetry you can beat the "theoretical maximum"- this tends to make rather poor solutions as it unbalances the cost function - sounds impossible? well it is just a technical matter really and not very interesting)
- Noise - the noise is measured to be in line with the theoretical minimum as discussed by [1]. That is the noise levels achieved are extremely good - the noise you end up with is unavoidable and results from the optical properties of the hologram itself - changing the optical setup is the only way to reduce the noise see [1].
- Practical use - use this algorithm to design your CGH, simulate it and then make it. If you make the hologram perfectly you will get exactly what you expect from the result of the design and simulation processes - no loss of quality or performance between design and realisation.

  Qualifiers - there are some physical limitations that the design algorithm is not aware of - use it outside of these (very wide limits) and things won't

be as perfect as claimed. Stick to the sampling guidelines and you'll be OK.

- Flexibility - you could adapt this algorithm to almost any optical setup- for instance it is perfectly possible to design reflection holograms or to put the hologram over a curved surface - you can also do things like multi-wavelength optimisation with this technique. However, all you get in this release is the code for designing single wavelength flat transmission holograms with the hologram elements laid out on a rectangular grid- if you absolutely must have the code for designing a hologram over an aspheric lens surface using fractal or Penrose tiling aperiodic pixel tiling you'll have to contact the author - I don't really expect to hear from anyone - these things while interesting aren't particularly useful and can't be made as far as I know.

3 Building the software

This software should be easy to build on any Unix platform. It has been built and used under Solaris, Cray-Unicos, Digital Unix / Tru64 and Linux. You will need a modern C++ compiler - I use g++ from gcc.

Steps to building-

1. Obtain and unpack the latest tar file in a sensible directory.
2. Build the software using make (type "make") - you may need to modify it which is easy because it is a very simple makefile.
3. copy the executables to wherever you put your executables - then you are all done.

4 Running the software

Assuming you have .con and .pos files ready the program is run from the command line like so `ds_expt myfile.con`, it generally takes a while especially if you have a big design so you'll probably want to run it in the background. If you have enabled reporting you can see the progress by looking in the report file.

Once the program has finished you will probably want to look at it using sim_ap or something similar (sim_ap should be available where you got this).

5 Control files

Each hologram requires a control (.con) file and one or more image position files (.pos). The format of the .pos files varies according to the type of image you want to design.

The control file is divided into three sections one for the algorithm and program, one for the hologram and one for the image. A typical control file looks like this

```
#-------------------------------------

options
        report          gasket_gs.rep 5000
endoptions

#-------------------------------------

hologram        gasket_gs.hol
        psize           0.000010 0.000010
        pixels          512 512
        phase           2
        wavelength      633e-9
        beamtype        0
        stable_at       .05
        circular_aperture
endhologram

#-------------------------------------

3dgsObject  gasket_gs.pos gasket_gs.amp
        imagesize       3513
end
endcon
```

5.1 Options section

This is virtually obsolete - the `report` command defines a file where the program dumps diagnostic data, the numerical argument defines how many trials (iterations) between data dumps. Use this if you wish to follow the progress of the program.

You can define cost parameters a and b here although they default to 2 and 1 which is a good generally setting.

The final option is `max_sweeps` which limits the number of complete hologram iterations (the number of times the whole hologram has been tested) - it defaults to 100. In practise the algorithm completes long before this, it is just there to prevent the program from using an infinite amount of computing time due to parameter errors.

5.2 Hologram section

This contains a large number of options - notice the name of the hologram file follows `hologram`.

- `psize <x> <y>` defines the size of the hologram pixels in metres.
- `pixels <x> <y>` defines the number of pixels in the hologram (if an aperture is imposed then the number used is less than x×y).

- `phase <levels>` defines the number of levels of phase permitted, 2 = binary phase hologram. The limitation on the number of phase levels is determined by CGH_TYPE which is currently char. This limits the number of levels to 128 - in practice run time is (roughly) proportional to the number of levels and more than 16 levels has very little effect on the quality and efficiency of the result.

- `wavelength <λ>` wavelength of the incident light in metres

- `beamtype <type>` This release supports uniform intensity (0) or Gaussian (1) - set the width with `beamdata <w>`. Other types are available but not included in this release.

- `stable_at <prob>` This defines when the algorithm stops. The algorithm stops when the probability of accepting a change falls below this value. This probability is estimate over a sample of 10000 trials - the error in this estimate is around 1% so values below 1% aren't really accurate (muck with samplesize in the source if you want). The measurement is mucked with because of the random exhaustive search technique and the true stats are very complicated. Rules of thumb - just trying it out use 0.05 - making a production quality design use 0.01.

  This setting affects the quality and efficiency of the design - see [2] for a better explanation of how it does this and what it costs in terms of optimisation time.

- `circular_aperture` makes the pixels inside of the biggest circle that fits into the gird of pixels active and those outside inactive. Without this the program uses all the pixels.

- `aperture_file` loads an aperture definition file - I can't see anyone using arbitrary apertures and I can't remember the format ask if you need it.

5.3 Image section

There are three image types supported in this release, 2dObject, 3dObject and 3dgsObject. Other types are available including phase and phase constrained objects, patches and pixel maps but I can't see anyone really wanting them as they are complicated and don't add much in the way of non-research advantages.

You can have any number of objects defined, I can't recall exactly what happens if you mix and match gs and non gs types (I think the non gs types default to their intensity being equal to the maximum intensities in the gs objects, I think all gs types match their maximum intensities - a small bug sorry).

Each follows a similar format- after the image type you get the .pos file name and the .amp filename. The optimisation results end up in the .amp file - you probably don't care about this though and will use a full reconstruction instead. The file format is ASCII and follows `real imaginary <CR>` in the same order as the .pos file.

The format of the .pos file varies from object to object, in these three cases they are all ASCII lists and you need to tell the object how long the list is with `imagesize`.

- `2dObject` you need to specify an additional parameter `distance` the distance from hologram to image (z) in metres.

  The file format is `x y<CR>`.

- `3dObject` The file format is `x y z<CR>`.

- `3dgsObject` The file format is `x y z I<CR>` where I is the relative intensity around the object.

6 Outputs

The hologram is saved as binary chars, one char per pixel.

The image amplitude data is saved as binary `complex <CGH_TYPE>`, (CGH_TYPE is currently defined as double).

Redefining the type to float *can* have a performance enhancing affect on some platforms, however error analysis and experiment suggests that numerical accuracy is an issue with CGHs larger than 128×128. On many platforms (for instance Alpha and Cray YMP8) changing to float actually slows things down - if you care benchmark first.

On a potentially interesting note the algorithm is robust in the face of these numerical errors- while there is a divergence of solutions due to the errors the algorithm still converges to a stable good solution. This is the basis behind a variation on the direct-search algorithm designed to give linear speed up for massively parallel architectures despite Amdahl's law.

7 "Strange" curves

If you look at the data produced in the report files and plot a few of the following graphs time (column 1) vs probability of accepting a change (column 2), time vs Cost A (or mean normalised intensity) (column 3) and time vs Cost B (or noise of mean normalised intensity) (column 4) you might be forgiven for thinking the graphs to produce strange curves.

7.1 Jumps and discontinuities

If efficiency and convergence time were not so important the hologram pixels would be chosen for trial at random - this gives two problems. (1) This would result in some pixels being reexamined soon after they had changed - since the solution would not have changed significantly the chance of the changing again would be very low. This phenomenon has been extensively studied and it was found that roughly 20 percent of all trials were null because of this (regardless of problem size) this leads to an increase in convergence time. (2) There is a finite probability that some pixels may never be tested - especially if the program is near optimal in computing efficiency - this leads to additional noise and slows the convergence down as these remaining pixels are randomly sought.

The solution is to use random-exhaustive selection (other non random exhaustive selection methods introduce artifacts and noise and slow down convergence - they can even prevent the solution converging). This algorithm implements random-exhaustive selection using a modified scrambled list procedure

which goes like this - a list of all the pixels is made, this is then randomly scrambled, the trials are then selected by going through the list. This in turn has a problem - the list requires far more memory than the hologram itself (8 times more) which is slow and very inefficient. The modified version generates a list for the X and Y coordinates of the pixels, the trials are then taken by chosing (x,y) pairs from the list. This obviously only generates N_x trials rather than $N_x \times N_y$ trials so the X list is rotated every N_x trials to generate $N_x \times N_y$ unique pairs. This "looks" random, it has pretty good random statistics and is fast and efficient.

Once the list has been used it is rescrambled - this prevents any fixed pattern noise associated with the list and it is this rescrambling that is responsible for the "strange" appearance of the graphs, without it (or with random selection) you get much smoother curves - in fact this smoothing out comes from the inclusion of a large number of unsuccesful trials associated with the decaying correlation of current solution with previous ones. The jumps you see occur when the list is rescrambled (or resused).

7.2 Hike in probability, flat spot in intensity and rising noise

At the second sweep (after the first rescrambling of the list) you will notice a large (temporary) increase in the probability of accepting a change. During this time the solution appears to get worse rather than better (Cost A stays flat and Cost B increases).

This occurs because the algorithm is forced to accept bad changes. To understand this you have to understand that the initial solution contains phase *and* amplitude information - accepting the bad changes removes the amplitude information and incurs the performance penalty you observe.

The amplitude information arises from the starting function used which determines the initial state of the solution. Traditionally this is a random phase distribution but this locks in noise and wastes time (because the initial image has to be calculated). In this program the optimization starts at the begining - the initial solution allows an additional state - pixel turned off. This eliminates the locked in noise associated with random (or fixed pattern) starts and saves time by eliminating the front end calculation of the initial solution. It has been shown to be worth at least 20 percent of the run time and to improve the reliability of convergence significantly.

At some point (early on) all the remaining off pixels must be forced on, this shows up as an increase in the probability of acceptance, rising noise - as you can see from the flat intensity during this time these off pixels contribute little to the solution.

It looks strange but this approach really does speed things up and more importantly it really does increase the performance of the algorithm (it reduces fixed pattern noise, opens additional routes through the solution space which enables faster convergence reduces the chance of being caught in a dodgy local solution).

8 License - GPL

References

[1] Matthew Clark. Two-dimensional, three-dimensional, and gray-scale images reconstructed from computer-generated holograms designed by use of a direct-search method. *Applied Optics*, 38(25):5331–5337, 1999.

[2] Matthew Clark and Robin Smith. A direct-search method for the computer design of holograms. *Optics Communications*, 124(1-2):150–164, 1996.