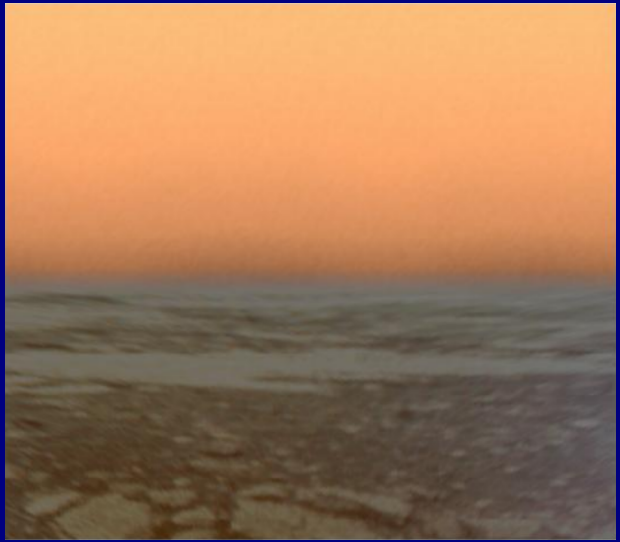
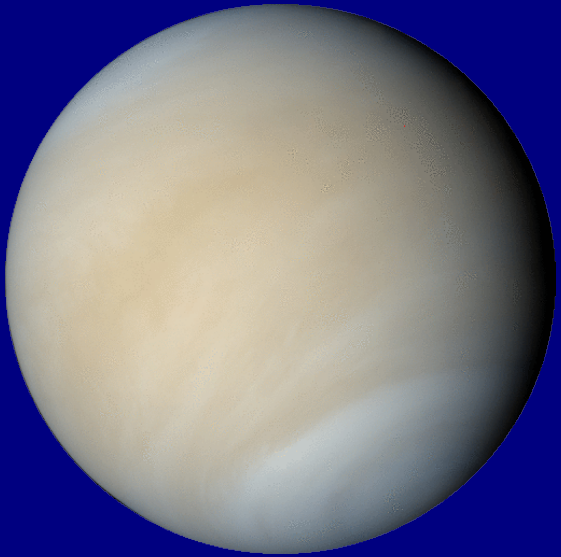
Life, Venus and Everything

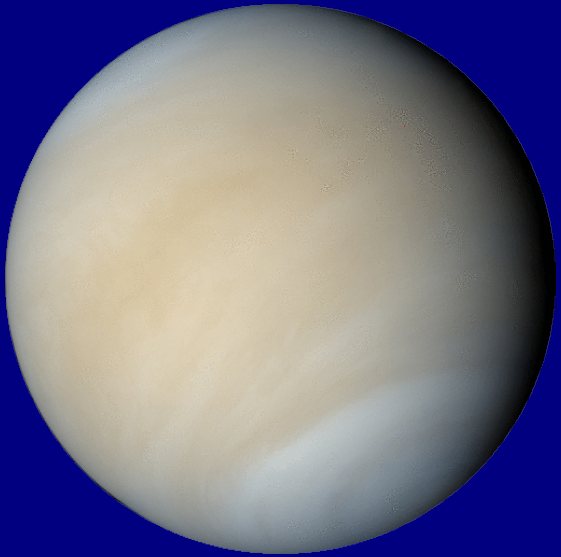
Björn Grieger (Aurora Technology B. V. for the European Space Agency, ESAC, Spain)



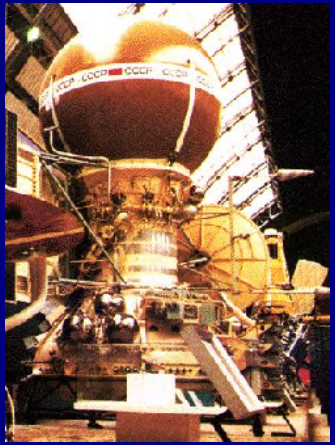
Venus in the visible



Venus in the visible

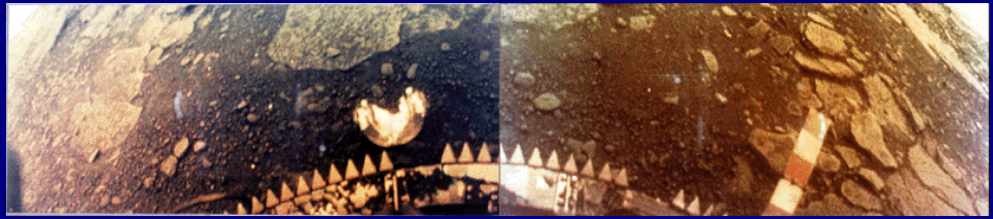


Venera 13 (1981–1982)

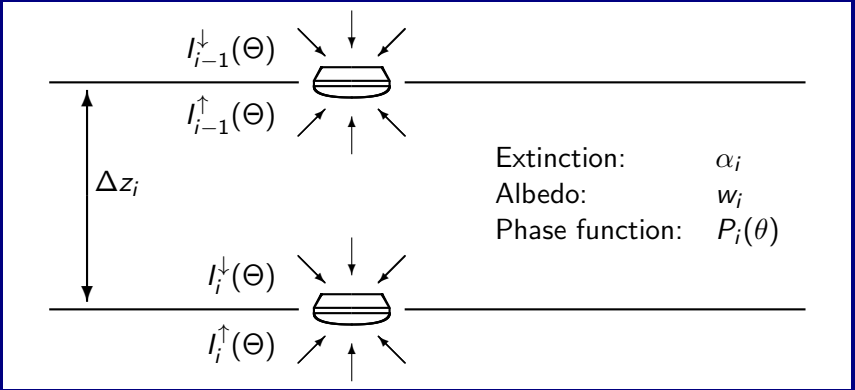


The lander survived more than two hours at 460°C and 90 bar.

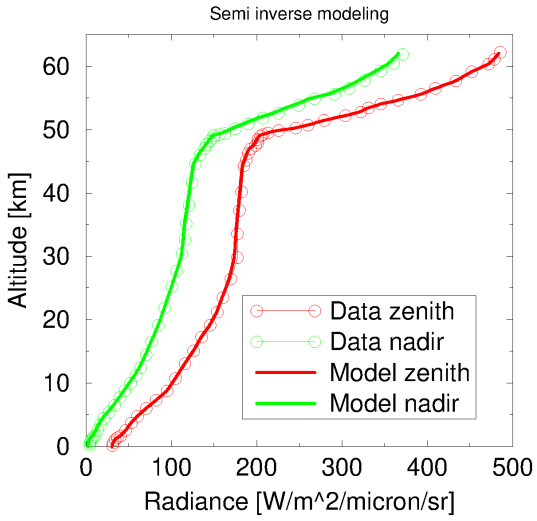
Venera 13 lander panorama



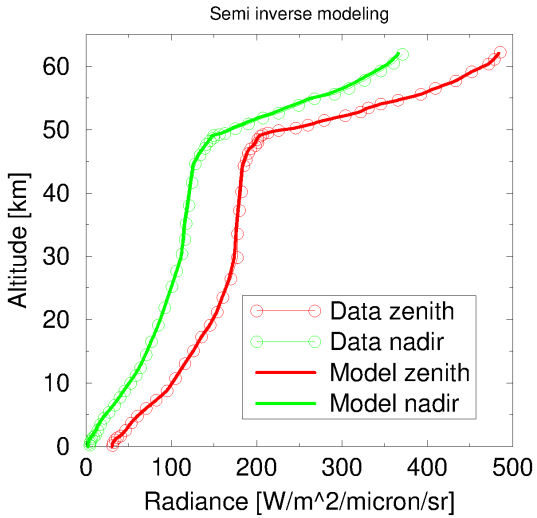
Titan Inverse Radiation Model (TIRM)



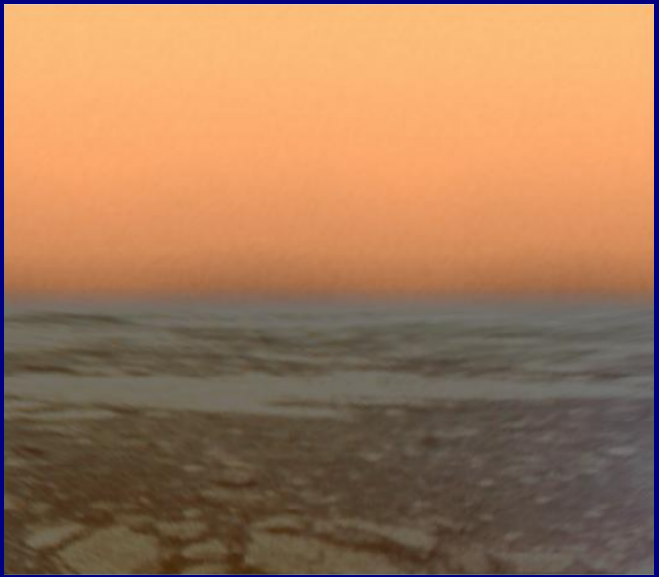
TIRM, Venus version



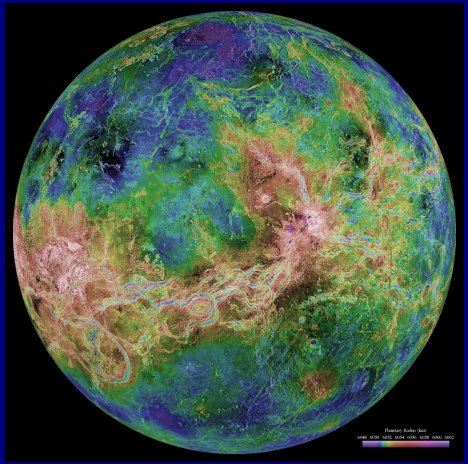
TIRM, Venus version



Surface from Venera 13 panorama + sky from TiRM



The Magellan radar mission



Magellan's death 1521 on the beach of Mactan island

The Death of Magellan

Magellan's death 1521 on the beach of Mactan island

Magellan's death 1521 on the beach of Mactan island

The Death of Magellan

Kampilan

Mera Espada

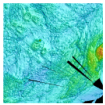
ESAC historical fencing club

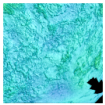
Magellan's death 1521 on the beach of Mactan island

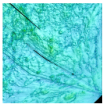
Kampilan

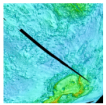
Kris

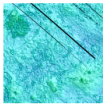
Hierarchical Progressive Survey (HiPS)

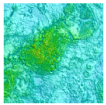

Npix0.jpg

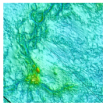

Npix1.jpg

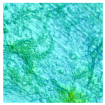

Npix2.jpg

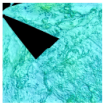

Npix3.jpg

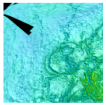

Npix4.jpg

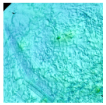

Npix5.jpg

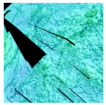

Npix6.jpg


Npix7.jpg


Npix8.jpg

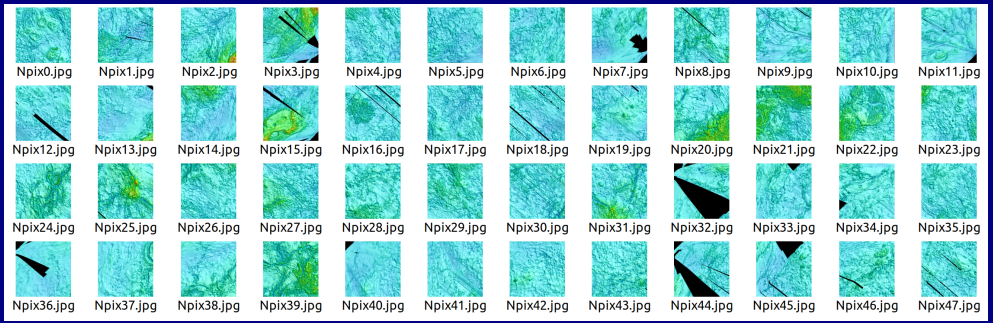

Npix9.jpg


Npix10.jpg


Npix11.jpg

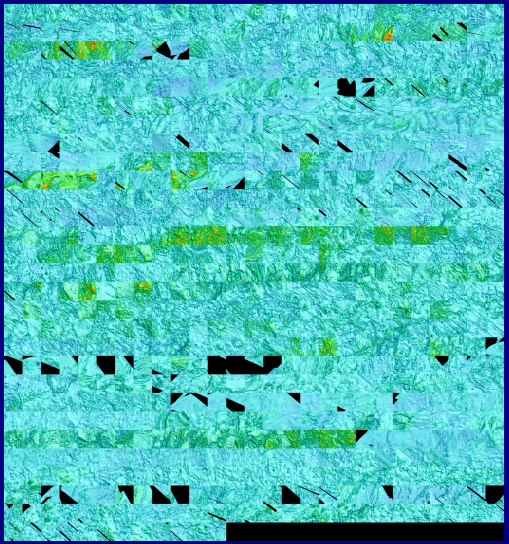
$N_{\text{order}} = 0$

Hierarchical Progressive Survey (HiPS)



$N_{\text{order}} = 1$

Hierarchical Progressive Survey (HiPS)



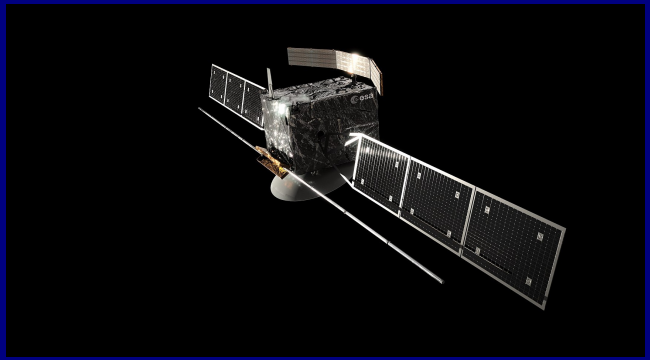
$N_{\text{order}} = 3$

Options for displaying HiPS

1. Locally with the application Aladin Desktop, or
2. embedded in a web page with Aladin Lite:

   http://comsim.esac.esa.int/rossim/bgrieger/VENUS

EnVision



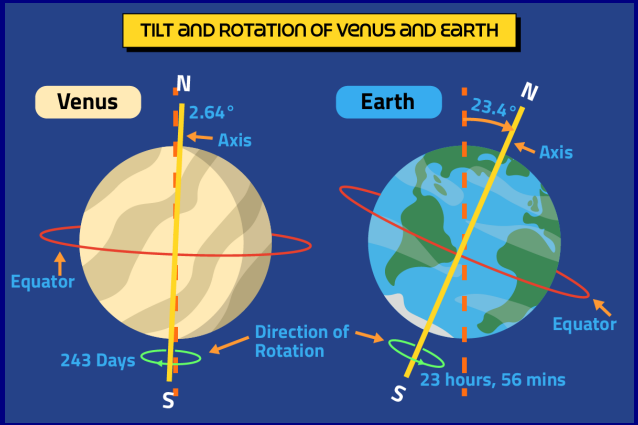
Intruments

- Venus Synthetic Aperture Radar (VenSAR)
- Venus Subsurface Radar Sounder (SRS)
- Venus Spectroscopy Suite (VenSpec)
- Radio Science Experiment

Orbit

- $1\frac{1}{2}$ hour period
- Near polar
- Inertially fixed orbit plane

Venus orbit and rotation

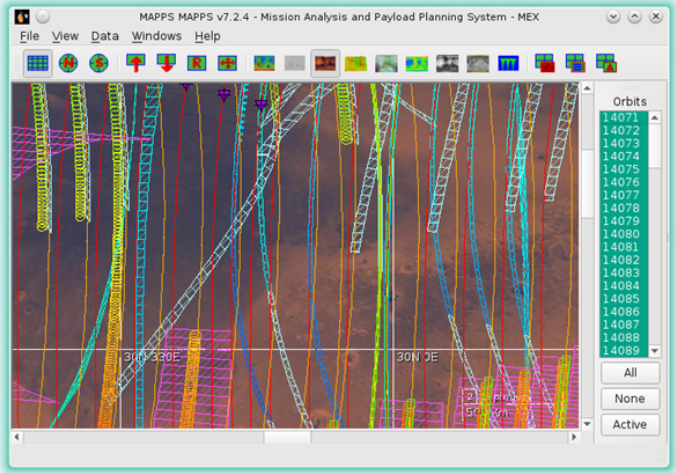


Venus year:	225 Earth days
Venus siderial day (1 cycle):	243 Earth days
Venus solar day:	117 Earth days
EnVision nominal mission:	6 cycles (4 Earth years)

Venus orbit and rotation

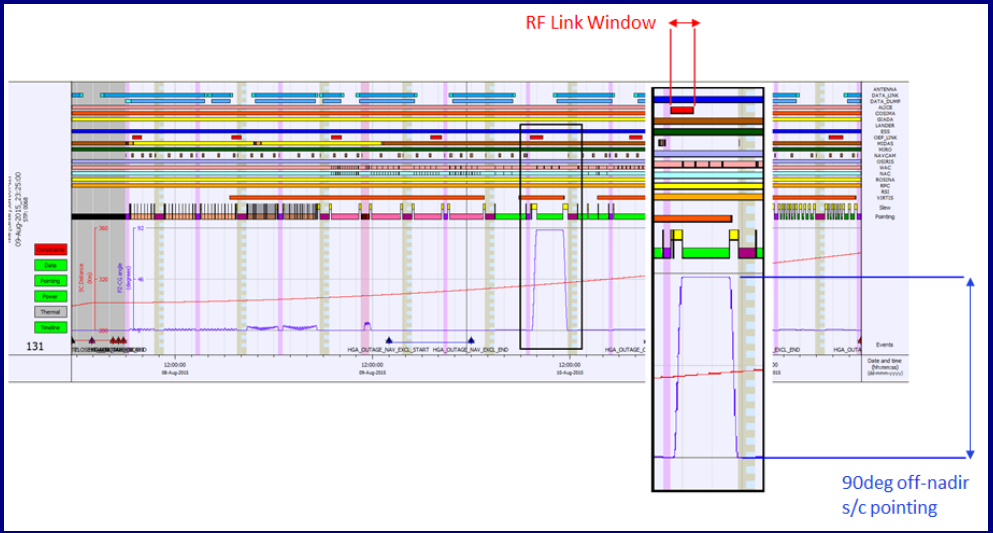
Note to self: show `maps.pdf`

Mission Analysis and Payload Planning System



Map with field of view overlay (Mars Express)

Mission Analysis and Payload Planning System



Timeline visualisation (Rosetta)

Big and strong or small and nimble?

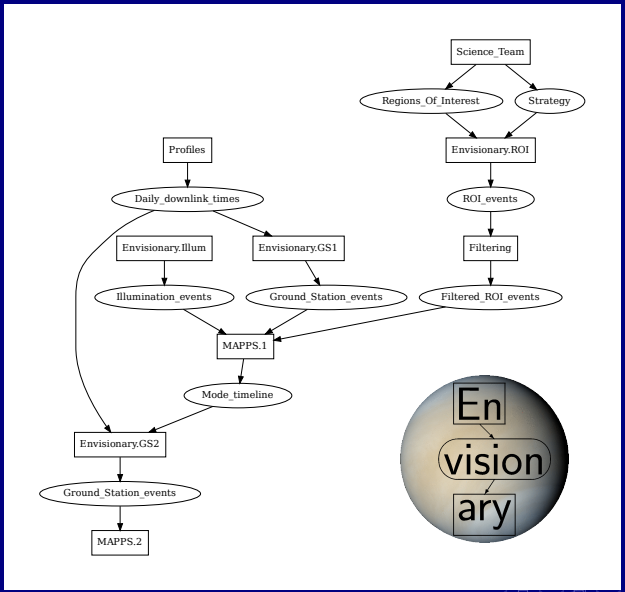
- MAPPS is used by several ESA planetary missions, some of these in operation.
- New developments and modifications have to be carefully planned, priority is given to missions in operation.
- MAPPS development cannot quickly react to EnVisions' needs (e. g., importing ROIs from files in weekly changing formats 😉).

Big and strong or small and nimble?

- MAPPS is used by several ESA planetary missions, some of these in operation.
- New developments and modifications have to be carefully planned, priority is given to missions in operation.
- MAPPS development cannot quickly react to EnVisions' needs (e. g., importing ROIs from files in weekly changing formats 😉).

$\Rightarrow$ Envisionary, a lightweight tool to supplement (the heavyweight) MAPPS with very specialized functionalities.

MAPPS and Envisionary



Functional(?) programming

- Developed in the 1950s, mostly used in academics, now big tech companies are picking it up.
- I was applying functional programming long before I first heard the phrase (and you probably, too).
- The name may not be quite elucidating to many:
  - All programming languages know *functions*, some — e. g. R — know nothing else.
  - The definition of *pure* functions and the lambda calculus used to describe functional programming are quite abstract.
- Describing the same thing in terms of data flow is much more intuitive.

Tools using data flow descriptions

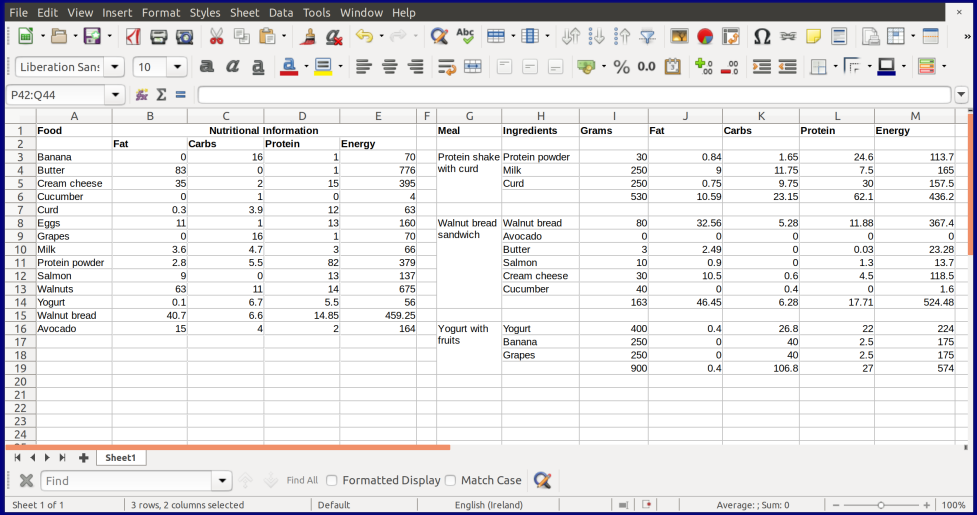
- (Spreadsheets)[1]
- OpenDX (aka IMB DataExplorer)
- The ∗nix make utility
- The arcs[2] wrapper language for make
- The dataflow[2] C++ template library

Note to self: skip to ▸ The ∗nix make utility !

[1] *Not* using data flow — only discussed to elucidate the not so obvious reason to call functional programming "functional".

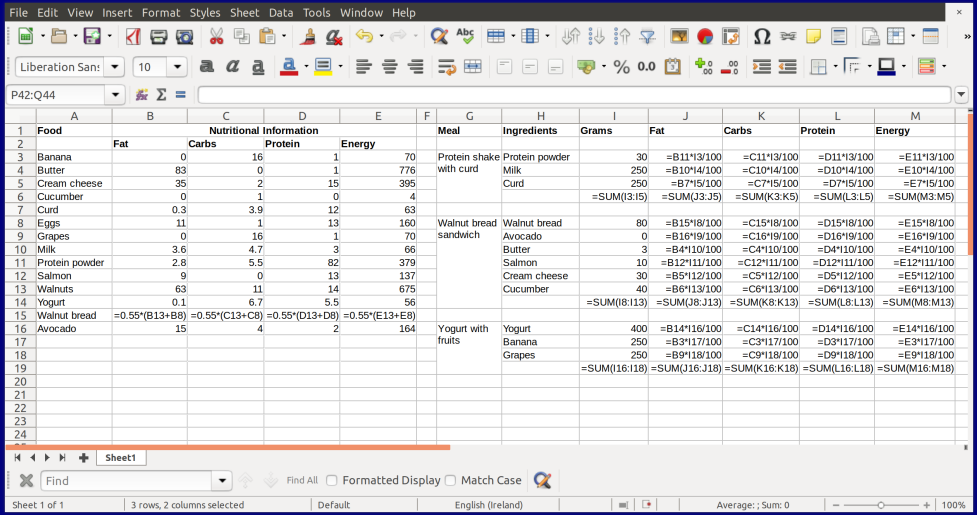
[2] Home grown.

A simple spreadsheet



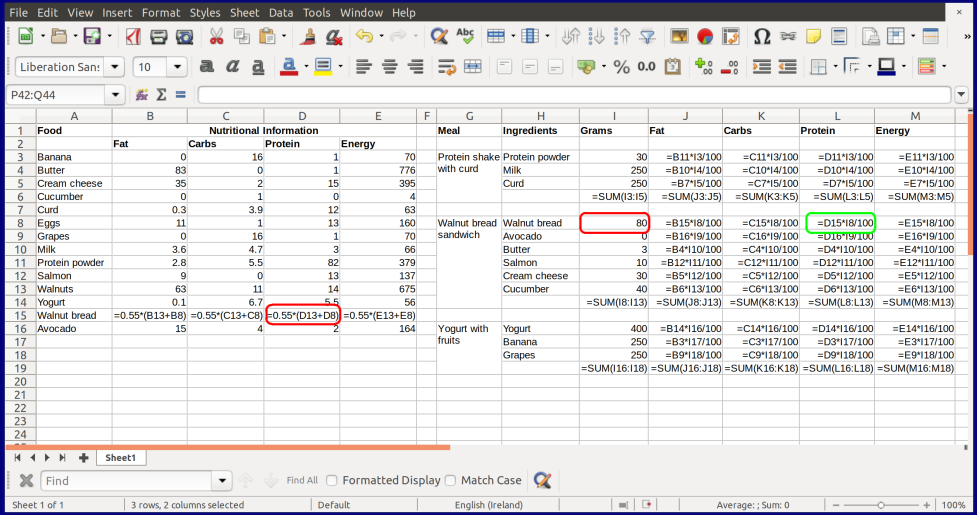
	A	B	C	D	E	F	G	H	I	J	K	L	M
1	**Food**	**Nutritional Information**					**Meal**	**Ingredients**	**Grams**	**Fat**	**Carbs**	**Protein**	**Energy**
2		**Fat**	**Carbs**	**Protein**	**Energy**								
3	Banana	0	16	1	70		Protein shake with curd	Protein powder	30	0.84	1.65	24.6	113.7
4	Butter	83	0	1	776			Milk	250	9	11.75	7.5	165
5	Cream cheese	35	2	15	395			Curd	250	0.75	9.75	30	157.5
6	Cucumber	0	1	0	4				530	10.59	23.15	62.1	436.2
7	Curd	0.3	3.9	12	63								
8	Eggs	11	1	13	160		Walnut bread sandwich	Walnut bread	80	32.56	5.28	11.88	367.4
9	Grapes	0	16	1	70			Avocado	0	0	0	0	0
10	Milk	3.6	4.7	3	66			Butter	3	2.49	0	0.03	23.28
11	Protein powder	2.8	5.5	82	379			Salmon	10	0.9	0	1.3	13.7
12	Salmon	9	0	13	137			Cream cheese	30	10.5	0.6	4.5	118.5
13	Walnuts	63	11	14	675			Cucumber	40	0	0.4	0	1.6
14	Yogurt	0.1	6.7	5.5	56				163	46.45	6.28	17.71	524.48
15	Walnut bread	40.7	6.6	14.85	459.25								
16	Avocado	15	4	2	164		Yogurt with fruits	Yogurt	400	0.4	26.8	22	224
17								Banana	250	0	40	2.5	175
18								Grapes	250	0	40	2.5	175
19									900	0.4	106.8	27	574
20													
21													
22													
23													
24													

A simple spreadsheet



	A	B	C	D	E	F	G	H	I	J	K	L	M
1	**Food**	**Nutritional Information**					**Meal**	**Ingredients**	**Grams**	**Fat**	**Carbs**	**Protein**	**Energy**
2		**Fat**	**Carbs**	**Protein**	**Energy**								
3	Banana	0	16	1	70		Protein shake with curd	Protein powder	30	=B11*I3/100	=C11*I3/100	=D11*I3/100	=E11*I3/100
4	Butter	83	0	1	776			Milk	250	=B10*I4/100	=C10*I4/100	=D10*I4/100	=E10*I4/100
5	Cream cheese	35	2	15	395			Curd	250	=B7*I5/100	=C7*I5/100	=D7*I5/100	=E7*I5/100
6	Cucumber	0	1	0	4				=SUM(I3:I5)	=SUM(J3:J5)	=SUM(K3:K5)	=SUM(L3:L5)	=SUM(M3:M5)
7	Curd	0.3	3.9	12	63								
8	Eggs	11	1	13	160		Walnut bread sandwich	Walnut bread	80	=B15*I8/100	=C15*I8/100	=D15*I8/100	=E15*I8/100
9	Grapes	0	16	1	70			Avocado	0	=B16*I9/100	=C16*I9/100	=D16*I9/100	=E16*I9/100
10	Milk	3.6	4.7	3	66			Butter	3	=B4*I10/100	=C4*I10/100	=D4*I10/100	=E4*I10/100
11	Protein powder	2.8	5.5	82	379			Salmon	10	=B12*I11/100	=C12*I11/100	=D12*I11/100	=E12*I11/100
12	Salmon	9	0	13	137			Cream cheese	30	=B5*I12/100	=C5*I12/100	=D5*I12/100	=E5*I12/100
13	Walnuts	63	11	14	675			Cucumber	40	=B6*I13/100	=C6*I13/100	=D6*I13/100	=E6*I13/100
14	Yogurt	0.1	6.7	5.5	56				=SUM(I8:I13)	=SUM(J8:J13)	=SUM(K8:K13)	=SUM(L8:L13)	=SUM(M8:M13)
15	Walnut bread	=0.55*(B13+B8)	=0.55*(C13+C8)	=0.55*(D13+D8)	=0.55*(E13+E8)								
16	Avocado	15	4	2	164		Yogurt with fruits	Yogurt	400	=B14*I16/100	=C14*I16/100	=D14*I16/100	=E14*I16/100
17								Banana	250	=B3*I17/100	=C3*I17/100	=D3*I17/100	=E3*I17/100
18								Grapes	250	=B9*I18/100	=C9*I18/100	=D9*I18/100	=E9*I18/100
19									=SUM(I16:I18)	=SUM(J16:J18)	=SUM(K16:K18)	=SUM(L16:L18)	=SUM(M16:M18)
20													
21													
22													
23													
24													

A simple spreadsheet



	A	B	C	D	E	F	G	H	I	J	K	L	M
1	**Food**	**Nutritional Information**					**Meal**	**Ingredients**	**Grams**	**Fat**	**Carbs**	**Protein**	**Energy**
2		**Fat**	**Carbs**	**Protein**	**Energy**								
3	Banana	0	16	1	70		Protein shake	Protein powder	30	=B11*I3/100	=C11*I3/100	=D11*I3/100	=E11*I3/100
4	Butter	83	0	1	776		with curd	Milk	250	=B10*I4/100	=C10*I4/100	=D10*I4/100	=E10*I4/100
5	Cream cheese	35	2	15	395			Curd	250	=B7*I5/100	=C7*I5/100	=D7*I5/100	=E7*I5/100
6	Cucumber	0	1	0	4				=SUM(I3:I5)	=SUM(J3:J5)	=SUM(K3:K5)	=SUM(L3:L5)	=SUM(M3:M5)
7	Curd	0.3	3.9	12	63								
8	Eggs	11	1	13	160		Walnut bread	Walnut bread	80	=B15*I8/100	=C15*I8/100	=D15*I8/100	=E15*I8/100
9	Grapes	0	16	1	70		sandwich	Avocado	0	=B16*I9/100	=C16*I9/100	=D16*I9/100	=E16*I9/100
10	Milk	3.6	4.7	3	66			Butter	3	=B4*I10/100	=C4*I10/100	=D4*I10/100	=E4*I10/100
11	Protein powder	2.8	5.5	82	379			Salmon	10	=B12*I11/100	=C12*I11/100	=D12*I11/100	=E12*I11/100
12	Salmon	9	0	13	137			Cream cheese	30	=B5*I12/100	=C5*I12/100	=D5*I12/100	=E5*I12/100
13	Walnuts	63	11	14	675			Cucumber	40	=B6*I13/100	=C6*I13/100	=D6*I13/100	=E6*I13/100
14	Yogurt	0.1	6.7	5.5	56				=SUM(I8:I13)	=SUM(J8:J13)	=SUM(K8:K13)	=SUM(L8:L13)	=SUM(M8:M13)
15	Walnut bread	=0.55*(B13+B8)	=0.55*(C13+C8)	=0.55*(D13+D8)	=0.55*(E13+E8)								
16	Avocado	15	4	2	164		Yogurt with	Yogurt	400	=B14*I16/100	=C14*I16/100	=D14*I16/100	=E14*I16/100
17							fruits	Banana	250	=B3*I17/100	=C3*I17/100	=D3*I17/100	=E3*I17/100
18								Grapes	250	=B9*I18/100	=C9*I18/100	=D9*I18/100	=E9*I18/100
19									=SUM(I16:I18)	=SUM(J16:J18)	=SUM(K16:K18)	=SUM(L16:L18)	=SUM(M16:M18)
20													
21													
22													
23													
24													

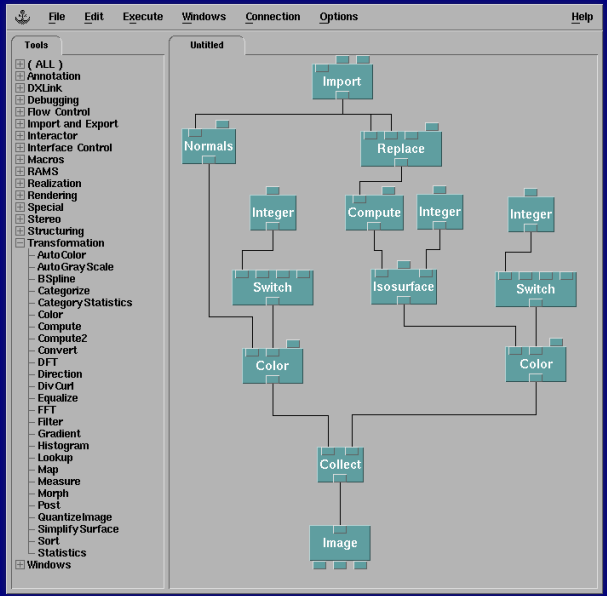
Functions and data flow

- A spreadsheet cell can contain a function of values from other cells (which may also contain functions).
- Such a function establishes a data flow between cells.
- Note that you don't write a sequence of commands (imperative programming), but you define a data flow (functional programming). The program decides what to execute, and when.
- A data flow *can* be described by functions, but it does not *have to.*

OpenDX

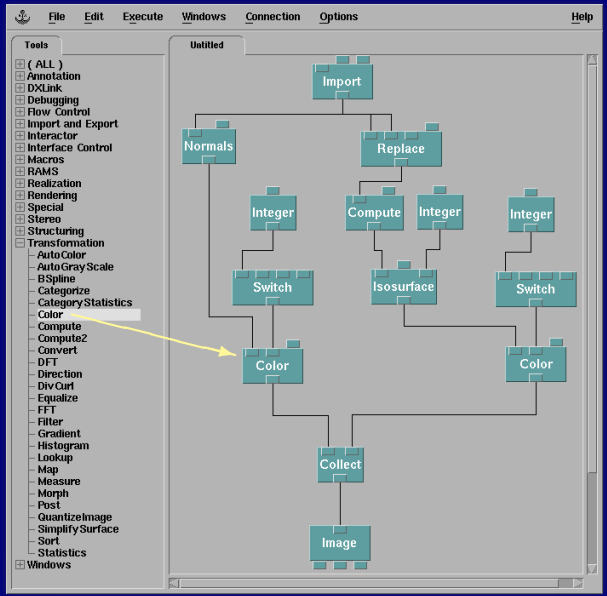
- Powerful 3D visualization system introduced by IBM in 1991 as Data Explorer, also promoted by other Unix platform vendors, particularly SGI.
- Envisaged to superseed IDL in the world of scientific visualization, but never really succeded. In 2000 handed over to the open source community as OpenDX. Further development idled out about 2007.
- True data-flow implementation, all modules are pure functions (i. e., their outputs are fully defined by their inputs). Hence, processes are stateless with no side effects.
- Uses (pure) functions under the hood but in the visual programming interface you directly create a data flow.

Visual Program Editor



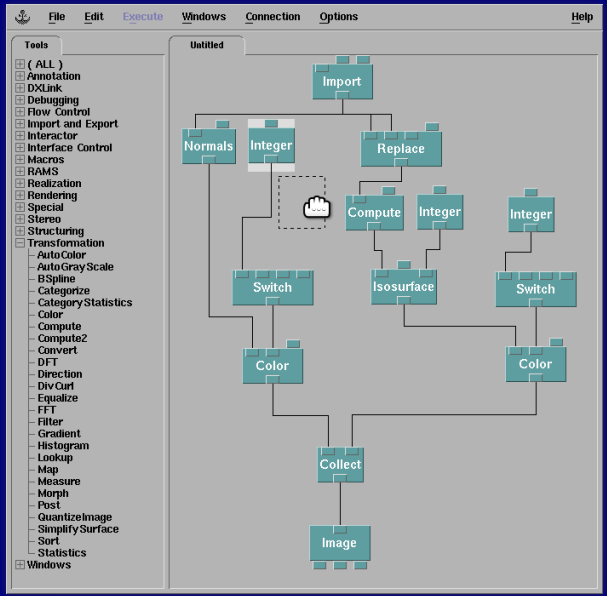
- A program is composed visually.

Visual Program Editor



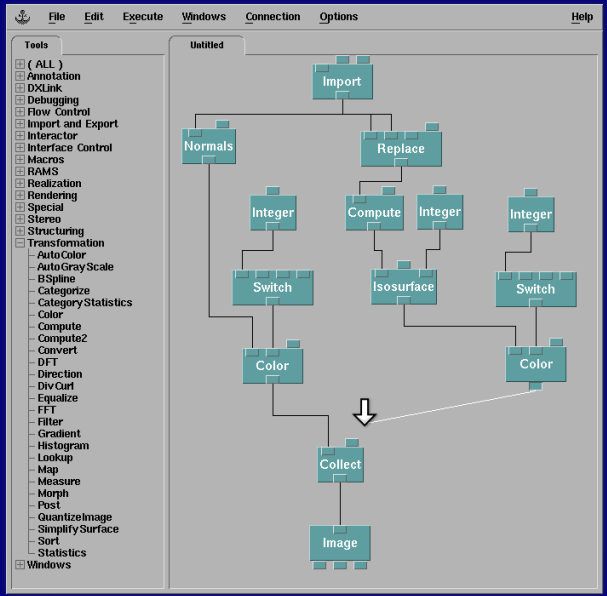
- A program is composed visually.
- Modules from the tool bar are placed on the canvas.

Visual Program Editor



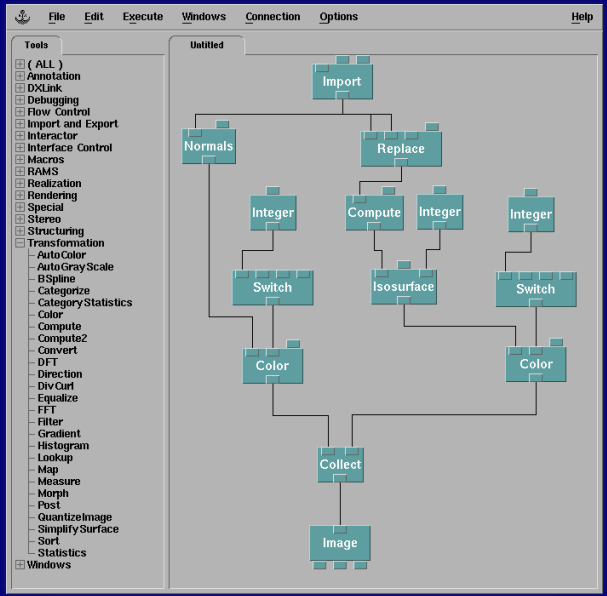
- A program is composed visually.
- Modules from the tool bar are placed on the canvas.
- Modules can be dragged around.

Visual Program Editor



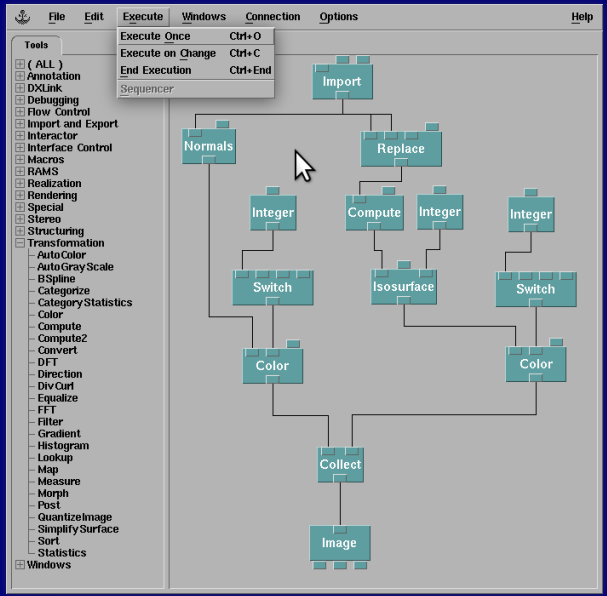
- A program is composed visually.
- Modules from the tool bar are placed on the canvas.
- Modules can be dragged around.
- Modules are connected by click and drag.

Visual Program Editor



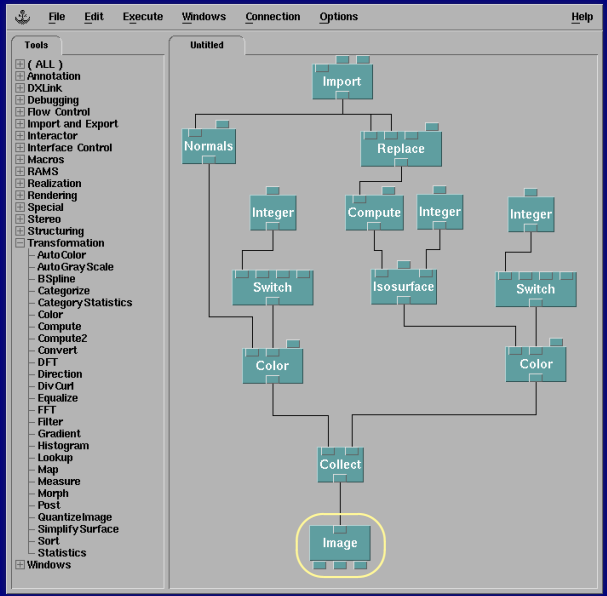
- A program is composed visually.
- Modules from the tool bar are placed on the canvas.
- Modules can be dragged around.
- Modules are connected by click and drag.
- The visual program defines a data flow.

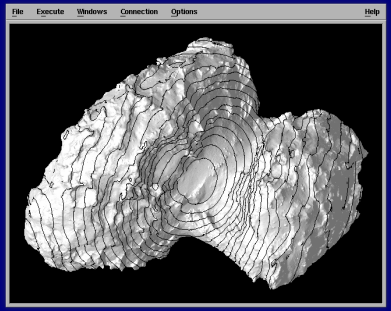
Visual Program Editor



- A program is composed visually.
- Modules from the tool bar are placed on the canvas.
- Modules can be dragged around.
- Modules are connected by click and drag.
- The visual program defines a data flow.
- It can be executed on demand or automatically if something changes.

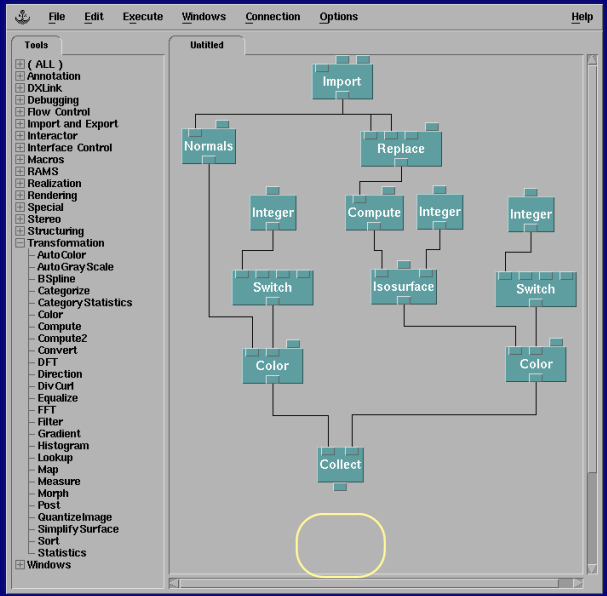
Execution model based on the data flow concept

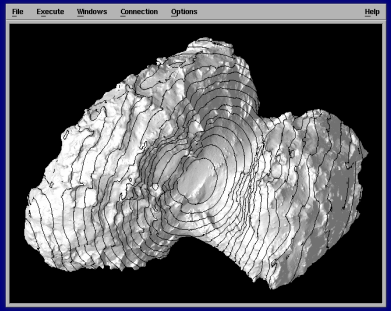




- When the program is executed, the **Image** module creates an image.

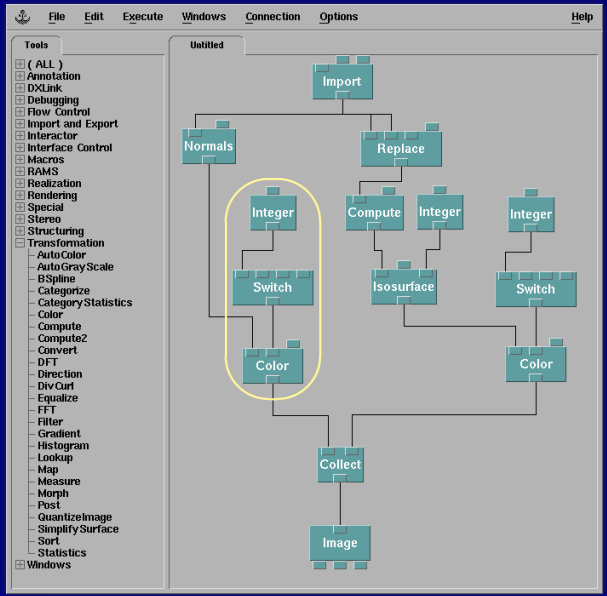
Execution model based on the data flow concept

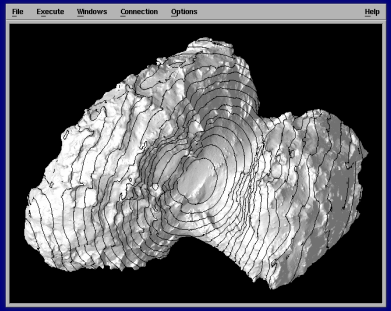




- When the program is executed, the **Image** module creates an image.
- If there was no **Image** module, the program would not execute at all!

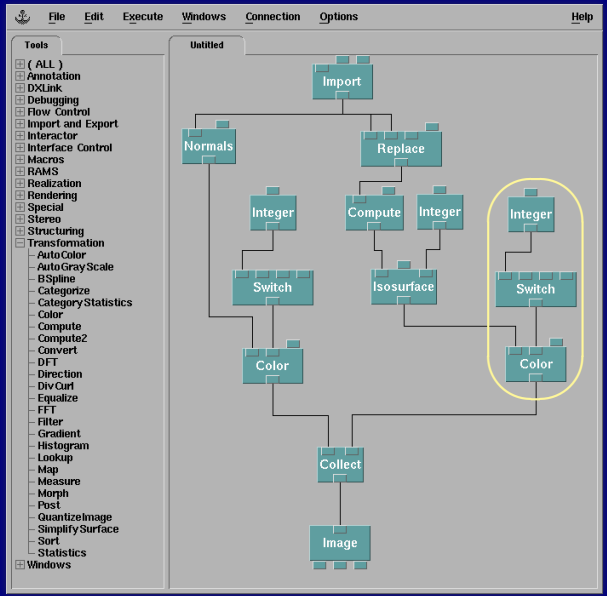
Execution model based on the data flow concept

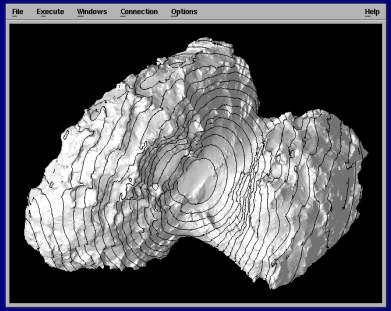




- This colors the shape (very quick).

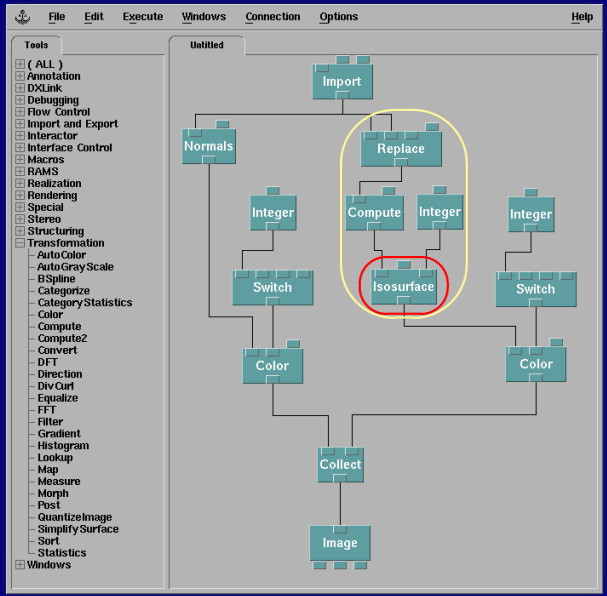
Execution model based on the data flow concept

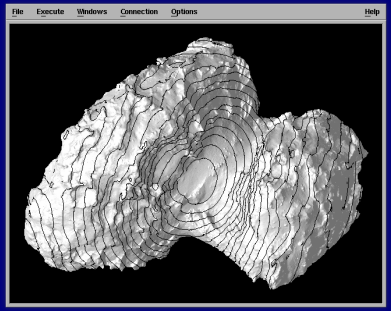




- This colors the shape (very quick).
- This colors the isolines (very quick).

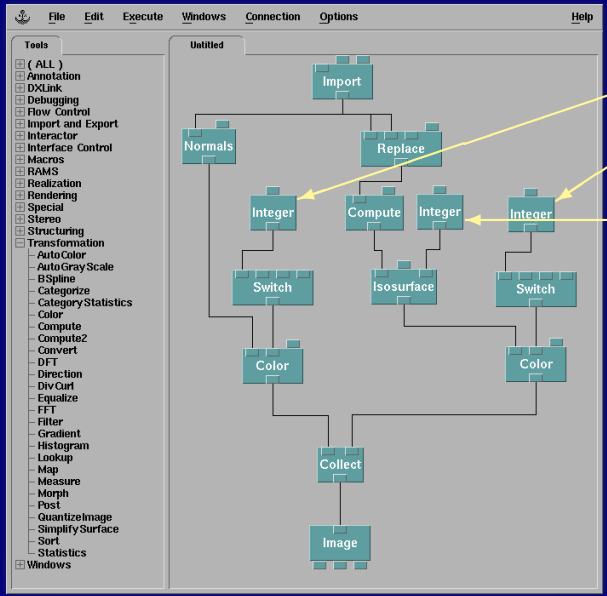
Execution model based on the data flow concept

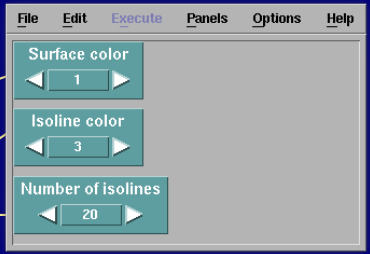




- This colors the shape (very quick).
- This colors the isolines (very quick).
- This creates the isolines (a bit time consuming).

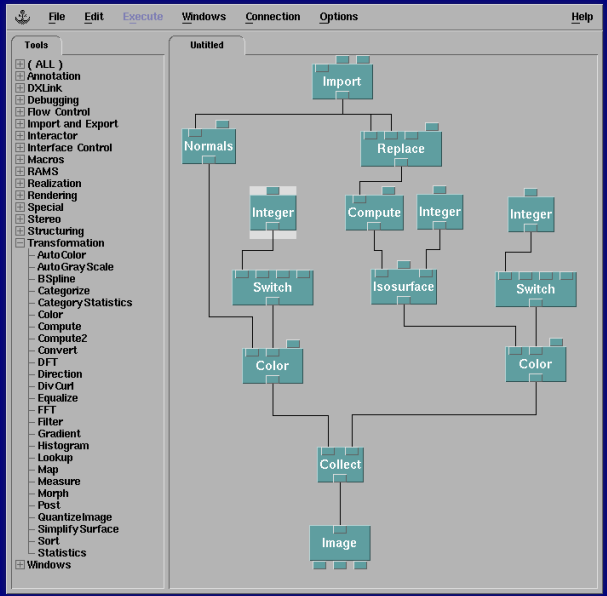
Execution model based on the data flow concept

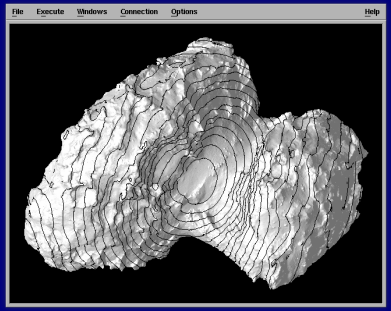




- So called “Interactors” allow to change input values interactively.

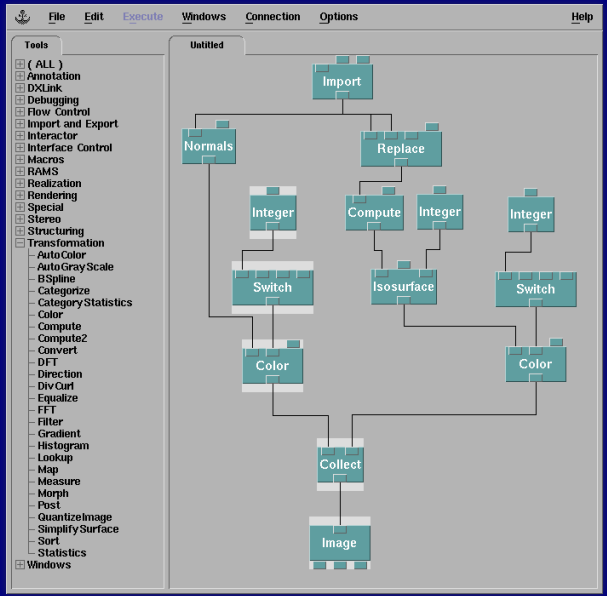
Execution model based on the data flow concept

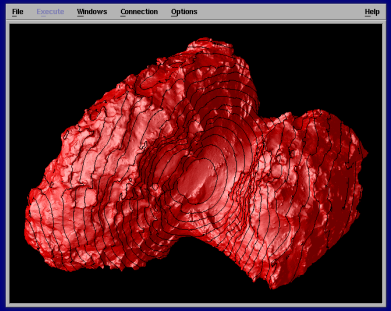




- If an input value, e. g., the surface color, is changed. . .

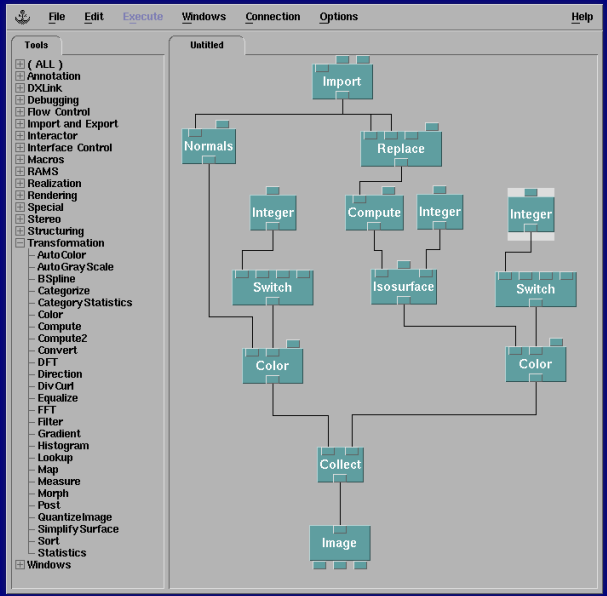
Execution model based on the data flow concept

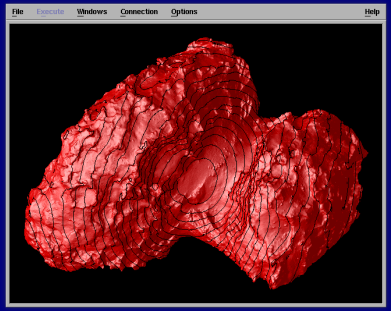




- If an input value, e. g., the surface color, is changed. . .
- . . . only downstream modules are re-executed — this is quick.

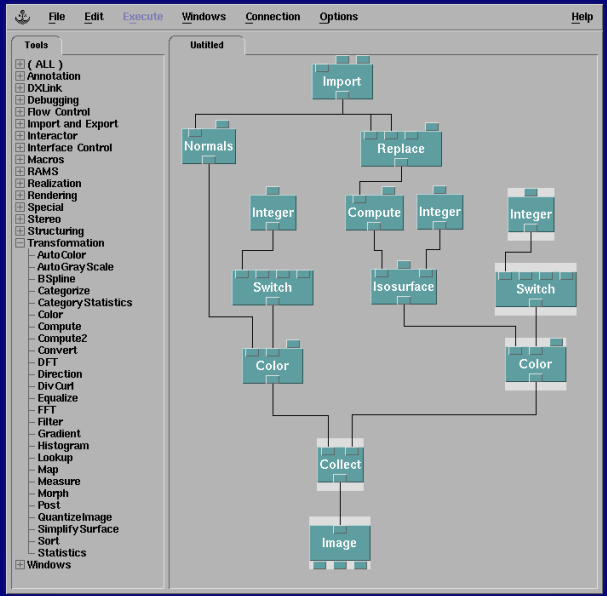
Execution model based on the data flow concept

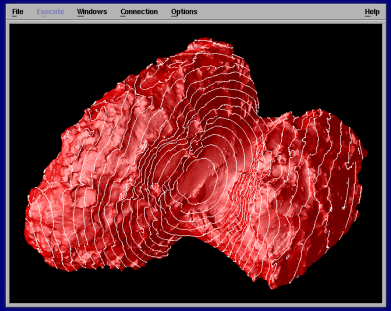




- Similarly, if the isoline color, is changed...

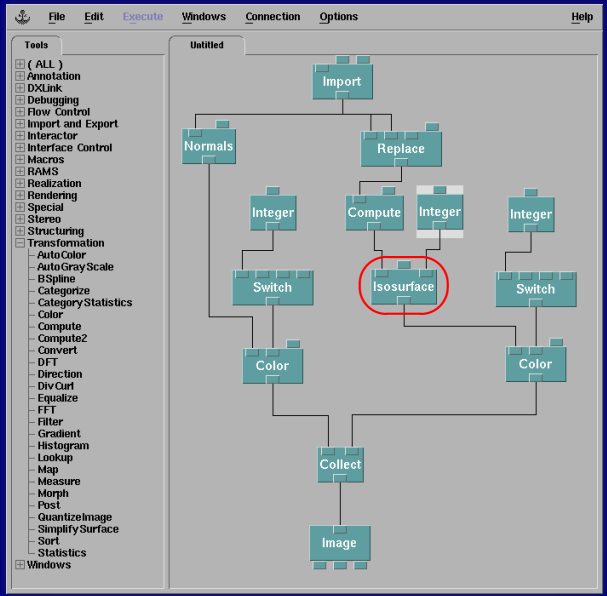
Execution model based on the data flow concept

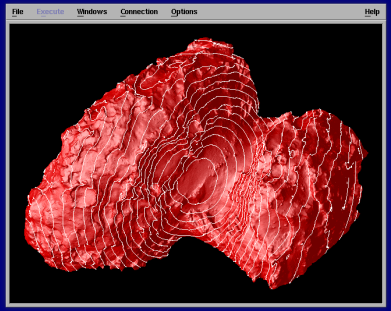




- Similarly, if the isoline color, is changed...
- ...it's quick.

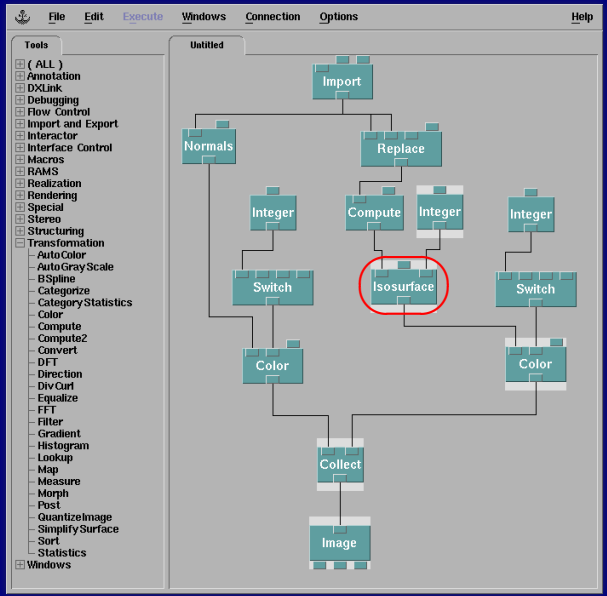
Execution model based on the data flow concept

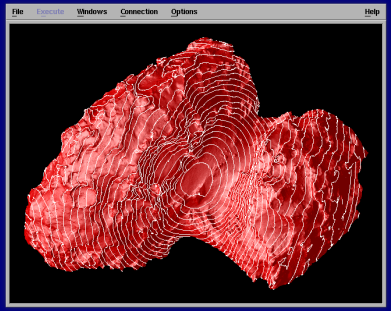




- Only if the isoline number is changed, the expensive Isosurface module has to be re-run...

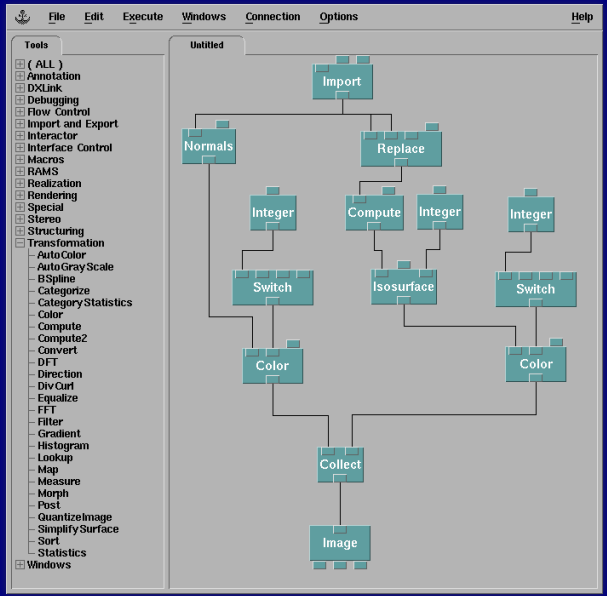
Execution model based on the data flow concept

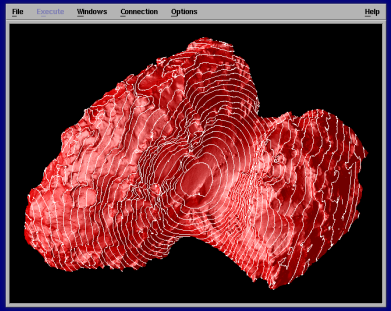




- Only if the isoline number is changed, the expensive Isosurface module has to be re-run...
- ...and takes a bit of computation time.

Execution model based on the data flow concept





- The data flow concept saves computation time. . .
- . . . and — even more important — development time!

The ∗nix make utility

```
make - GNU make utility to maintain groups of programs
```

- Traditionally used to build executables.
- But can be used for all kinds of computations.
- Basic building blocks are **rules** (though there is much more).

Rule

	Name	Content	Examples
What to make?			
What's needed?			
How to make it?			

Rule

	Name	Content	Examples
What to make?	Targets	List of files to make	Executables, object code files
What's needed?			
How to make it?			

Rule

	Name	Content	Examples
What to make?	Targets	List of files to make	Executables, object code files
What's needed?	Dependencies	List of files needed as input	Object code files, source code files
How to make it?			

Rule

	Name	Content	Examples
What to make?	Targets	List of files to make	Executables, object code files
What's needed?	Dependencies	List of files needed as input	Object code files, source code files
How to make it?	Recipe	Sequence of commands to execute	Compile and link commands

Note to self: skip to ▸ Orchestrating computations **!**

Makefile for traditional executable building

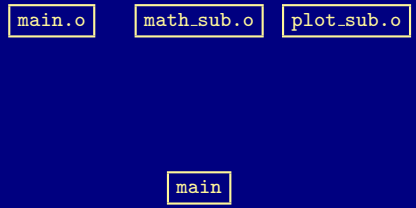
```
#  Link object files to create executable
#  --------------------------------------
#  Target
       main : \
#  Dependencies
       main.o math_sub.o plot_sub.o ; \
#  Recipe
       gfortran -o main main.o math_sub.o plot_sub.o

#  Compile main program
#  --------------------
#  Target
       main.o : \
#  Dependency
       main.f ; \
#  Recipe
       gfortran -c main.f

#  Compile math subroutines
#  ------------------------
#  Target
       math_sub.o : \
#  Dependency
       math_sub.f ; \
#  Recipe
       gfortran -c math_sub.f

#  Compile plot subroutines
#  ------------------------
#  Target
       plot_sub.o : \
#  Dependency
       plot_sub.f ; \
#  Recipe
       gfortran -c plot_sub.f
```

Makefile for traditional executable building

```
#  Link object files to create executable
#  --------------------------------------
#  Target
       main: \
#  Dependencies
       main.o math_sub.o plot_sub.o ; \
#  Recipe
       gfortran -o main main.o math_sub.o plot_sub.o

#  Compile main program
#  --------------------
#  Target
       main.o : \
#  Dependency
       main.f ; \
#  Recipe
       gfortran -c main.f

#  Compile math subroutines
#  ------------------------
#  Target
       math_sub.o : \
#  Dependency
       math_sub.f ; \
#  Recipe
       gfortran -c math_sub.f

#  Compile plot subroutines
#  ------------------------
#  Target
       plot_sub.o : \
#  Dependency
       plot_sub.f ; \
#  Recipe
       gfortran -c plot_sub.f
```

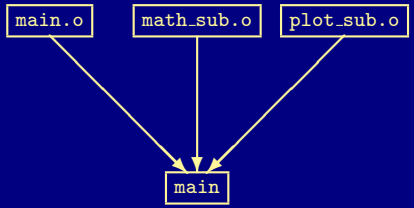
Makefile for traditional executable building

```
#  Link object files to create executable
#  --------------------------------------
#  Target
       main : \
#  Dependencies
       main.o math_sub.o plot_sub.o ; \
#  Recipe
       gfortran -o main main.o math_sub.o plot_sub.o

#  Compile main program
#  --------------------
#  Target
       main.o : \
#  Dependency
       main.f ; \
#  Recipe
       gfortran -c main.f

#  Compile math subroutines
#  ------------------------
#  Target
       math_sub.o : \
#  Dependency
       math_sub.f ; \
#  Recipe
       gfortran -c math_sub.f

#  Compile plot subroutines
#  ------------------------
#  Target
       plot_sub.o : \
#  Dependency
       plot_sub.f ; \
#  Recipe
       gfortran -c plot_sub.f
```



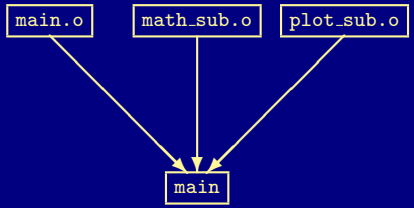
Makefile for traditional executable building

```
#  Link object files to create executable
#  --------------------------------------
#  Target
       main : \
#  Dependencies
       main.o math_sub.o plot_sub.o ; \
#  Recipe
       gfortran -o main main.o math_sub.o plot_sub.o

#  Compile main program
#  --------------------
#  Target
       main.o : \
#  Dependency
       main.f ; \
#  Recipe
       gfortran -c main.f

#  Compile math subroutines
#  ------------------------
#  Target
       math_sub.o : \
#  Dependency
       math_sub.f ; \
#  Recipe
       gfortran -c math_sub.f

#  Compile plot subroutines
#  ------------------------
#  Target
       plot_sub.o : \
#  Dependency
       plot_sub.f ; \
#  Recipe
       gfortran -c plot_sub.f
```



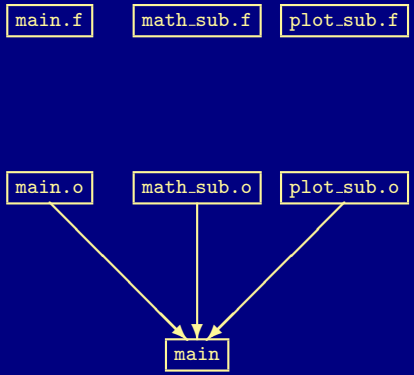
Makefile for traditional executable building

```
#  Link object files to create executable
#  --------------------------------------
#  Target
       main : \
#  Dependencies
       main.o math_sub.o plot_sub.o ; \
#  Recipe
       gfortran -o main main.o math_sub.o plot_sub.o

#  Compile main program
#  --------------------
#  Target
       main.o : \
#  Dependency
       main.f ; \
#  Recipe
       gfortran -c main.f

#  Compile math subroutines
#  ------------------------
#  Target
       math_sub.o : \
#  Dependency
       math_sub.f ; \
#  Recipe
       gfortran -c math_sub.f

#  Compile plot subroutines
#  ------------------------
#  Target
       plot_sub.o : \
#  Dependency
       plot_sub.f ; \
#  Recipe
       gfortran -c plot_sub.f
```



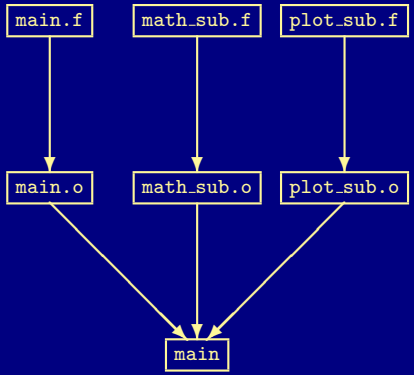
Makefile for traditional executable building

```
#  Link object files to create executable
#  --------------------------------------
#  Target
       main : \
#  Dependencies
       main.o math_sub.o plot_sub.o ; \
#  Recipe
       gfortran -o main main.o math_sub.o plot_sub.o

#  Compile main program
#  --------------------
#  Target
       main.o : \
#  Dependency
       main.f ; \
#  Recipe
       gfortran -c main.f

#  Compile math subroutines
#  ------------------------
#  Target
       math_sub.o : \
#  Dependency
       math_sub.f ; \
#  Recipe
       gfortran -c math_sub.f

#  Compile plot subroutines
#  ------------------------
#  Target
       plot_sub.o : \
#  Dependency
       plot_sub.f ; \
#  Recipe
       gfortran -c plot_sub.f
```



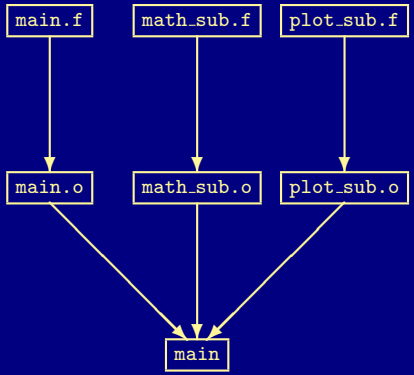
Makefile for traditional executable building

```
#  Link object files to create executable
#  --------------------------------------
#  Target
       main : \
#  Dependencies
       main.o math_sub.o plot_sub.o ; \
#  Recipe
       gfortran -o main main.o math_sub.o plot_sub.o

#  Compile main program
#  --------------------
#  Target
       main.o : \
#  Dependency
       main.f ; \
#  Recipe
       gfortran -c main.f

#  Compile math subroutines
#  ------------------------
#  Target
       math_sub.o : \
#  Dependency
       math_sub.f ; \
#  Recipe
       gfortran -c math_sub.f

#  Compile plot subroutines
#  ------------------------
#  Target
       plot_sub.o : \
#  Dependency
       plot_sub.f ; \
#  Recipe
       gfortran -c plot_sub.f
```



Makefile for traditional executable building

```
#  Link object files to create executable
#  --------------------------------------
#  Target
       main : \
#  Dependencies
       main.o math_sub.o plot_sub.o ; \
#  Recipe
       gfortran -o main main.o math_sub.o plot_sub.o

#  Compile main program
#  --------------------
#  Target
       main.o : \
#  Dependency
       main.f ; \
#  Recipe
       gfortran -c main.f

#  Compile math subroutines
#  ------------------------
#  Target
       math_sub.o : \
#  Dependency
       math_sub.f ; \
#  Recipe
       gfortran -c math_sub.f

#  Compile plot subroutines
#  ------------------------
#  Target
       plot_sub.o : \
#  Dependency
       plot_sub.f ; \
#  Recipe
       gfortran -c plot_sub.f
```



The make syntax defines a data flow. It represents functional programming **without** *functions.*

Changing rules from building executables. . .

	Name	Content	Examples
What to make?	Targets	List of files to make	Executables, object code files
What's needed?	Dependencies	List of files needed as input	Object code files, source code files
How to make it?	Recipe	Sequence of commands to execute	Compile and link commands

Changing rules from building executables. . .

	Name	Content	Examples
What to make?	Targets	List of files to make	Output data files
What's needed?	Dependencies	List of files needed as input	Object code files, source code files
How to make it?	Recipe	Sequence of commands to execute	Compile and link commands

Changing rules from building executables...

	Name	Content	Examples
What to make?	Targets	List of files to make	Output data files
What's needed?	Dependencies	List of files needed as input	Input data files, program file
How to make it?	Recipe	Sequence of commands to execute	Compile and link commands

For conciseness, we assume here that the programs are interpreted and no compilation is needed.

Changing rules from building executables. . .

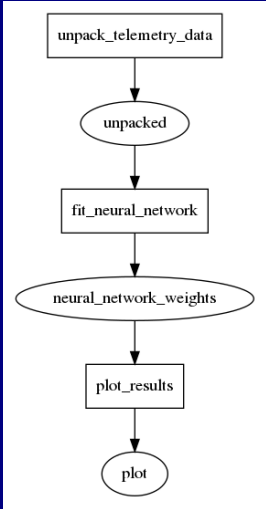
	Name	Content	Examples
What to make?	Targets	List of files to make	Output data files
What's needed?	Dependencies	List of files needed as input	Input data files, program file
How to make it?	Recipe	Sequence of commands to execute	Program call

Changing rules from building executables. . .

	Name	Content	Examples
What to make?	Targets	List of files to make	Output data files
What's needed?	Dependencies	List of files needed as input	Input data files, program file
How to make it?	Recipe	Sequence of commands to execute	Program call

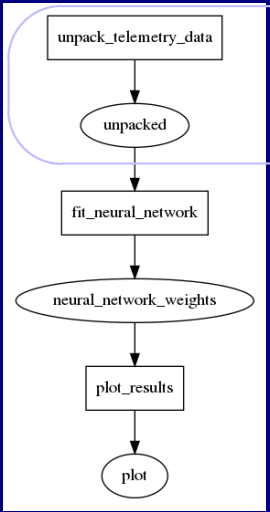
. . . to orchestrating computations.

Makefile for computation orchestration



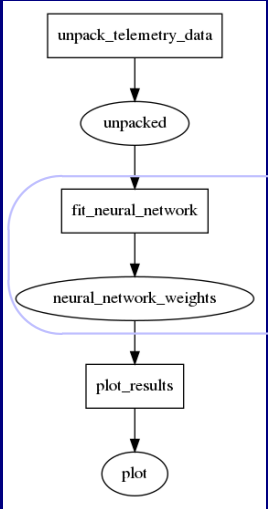

```
#   Unpack telemetry data
#   ---------------------
#   Target
        unpacked.dat : \
#   Dependencies
        unpack_telemetry_data.py ; \
#   Recipe
        ./unpack_telemetry_data.py

#   Fit neural network (very time consuming)
#   ----------------------------------------
#   Target
        neural_network_weights.dat : \
#   Dependency
        unpacked.dat fit_neural_network.py ; \
#   Recipe
        ./fit_neural_network.py

#   Plot results
#   ------------
#   Target
        plot.png : \
#   Dependency
        neural_network_weights.dat plot.py ; \
#   Recipe
        ./plot_results.py
```

Makefile for computation orchestration



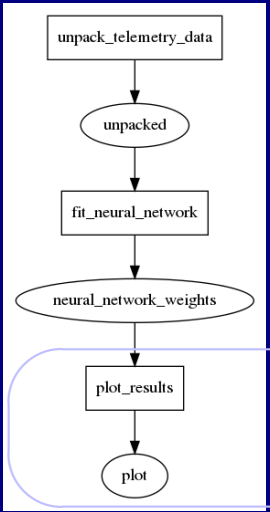

```
#  Unpack telemetry data
#  ---------------------
#  Target
       unpacked.dat : \
#  Dependencies
       unpack_telemetry_data.py ; \
#  Recipe
       ./unpack_telemetry_data.py

#  Fit neural network (very time consuming)
#  ----------------------------------------
#  Target
       neural_network_weights.dat : \
#  Dependency
       unpacked.dat fit_neural_network.py ; \
#  Recipe
       ./fit_neural_network.py

#  Plot results
#  ------------
#  Target
       plot.png : \
#  Dependency
       neural_network_weights.dat plot.py ; \
#  Recipe
       ./plot_results.py
```

Makefile for computation orchestration



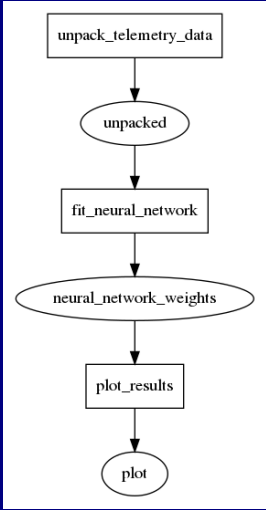

```
#  Unpack telemetry data
#  ---------------------
#  Target
       unpacked.dat : \
#  Dependencies
       unpack_telemetry_data.py ; \
#  Recipe
       ./unpack_telemetry_data.py

#  Fit neural network (very time consuming)
#  ----------------------------------------
#  Target
       neural_network_weights.dat : \
#  Dependency
       unpacked.dat fit_neural_network.py ; \
#  Recipe
       ./fit_neural_network.py

#  Plot results
#  ------------
#  Target
       plot.png : \
#  Dependency
       neural_network_weights.dat plot.py ; \
#  Recipe
       ./plot_results.py
```

Makefile for computation orchestration



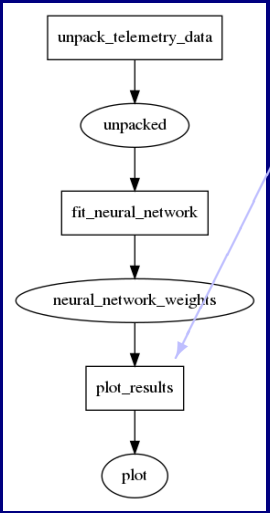

```
#  Unpack telemetry data
#  ---------------------
#  Target
       unpacked.dat : \
#  Dependencies
       unpack_telemetry_data.py ; \
#  Recipe
       ./unpack_telemetry_data.py

#  Fit neural network (very time consuming)
#  ----------------------------------------
#  Target
       neural_network_weights.dat : \
#  Dependency
       unpacked.dat fit_neural_network.py ; \
#  Recipe
       ./fit_neural_network.py

#  Plot results
#  ------------
#  Target
       plot.png : \
#  Dependency
       neural_network_weights.dat plot.py ; \
#  Recipe
       ./plot_results.py
```

Makefile for computation orchestration



The Makefile establishes a data flow between computations.

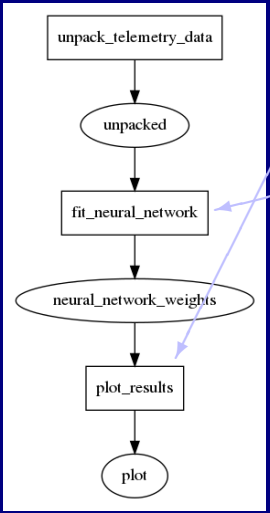
Makefile for computation orchestration



The Makefile establishes a data flow between computations.

If we change a color in the plot...

Makefile for computation orchestration

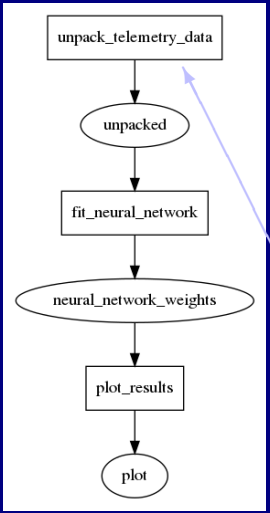


The Makefile establishes a data flow between computations.

If we change a color in the plot...

...we don't have to rerun the (expensive) fitting (and make won't).

Makefile for computation orchestration



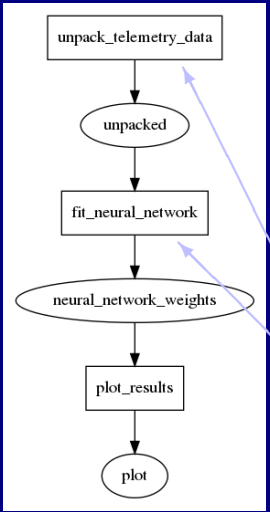
The Makefile establishes a data flow between computations.

If we change a color in the plot...

...we don't have to rerun the (expensive) fitting (and make won't).

If we correct a bug in the unpacking...

Makefile for computation orchestration



The Makefile establishes a data flow between computations.

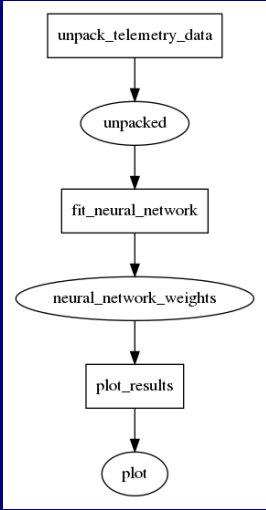
If we change a color in the plot...

...we don't have to rerun the (expensive) fitting (and make won't).

If we correct a bug in the unpacking...

...we *do* have to rerun the fitting (and the plotting), and make *will*.

Makefile for computation orchestration



The Makefile establishes a data flow between computations.

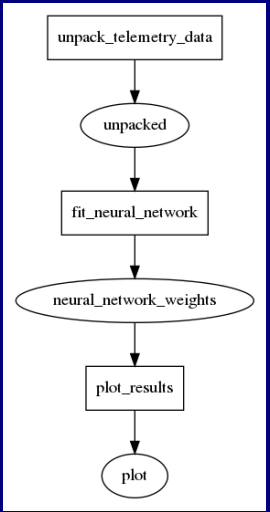
To have, e. g., `neural_network_weights` passed over from `fit_neural_network` to `plot_results` requires three steps:

1. Insert a rule in the Makefile.
2. Insert code to write the file `neural_network_weights.dat` in `fit_neural_network.py`.
3. Insert code to read the file in `plot_results.py`.

This is tedious and error prone.

Can we do better?

The arcs wrapper language for make

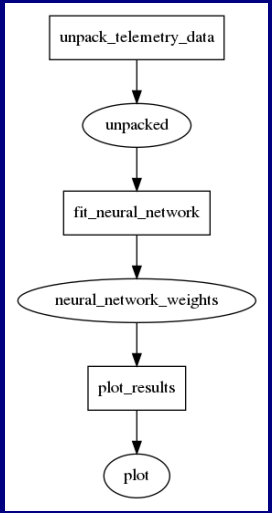


Idea:

- Describe the data flow in a file (with an appropriate language).
- Have a compiler reading that file and
  1. writing a Makefile,
  2. inserting code to write respective files into the source code of all programs which output data,
  3. inserting code to read respective files into the source code of all programs which input data.

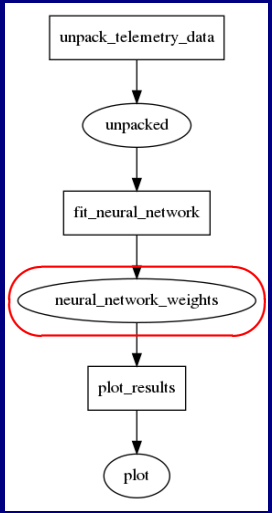
Contrary to make, there is now a single point of maintenance for the "pathways of data between programs" which we simply call arcs — and likewise the language and the compiler.

The arcs wrapper language for make



- In 2008, I coded continuously over one weekend, almost without eat, drink, sleep. After that, I had a basically working compiler.
- Since then, it was constantly developed “on the job”. Now it has about 1000 lines of code (2000 with comments and white space).
- The invested time was minimal, so it was written quick and dirty, which has resulted in horrendous spaghetti code.

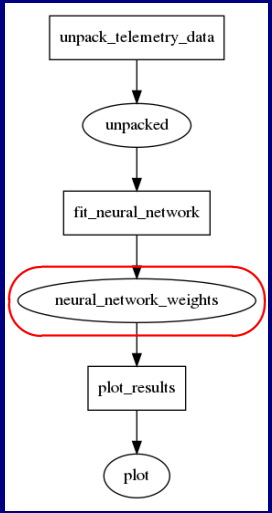
The arcs wrapper language for make




```
\neural_network_weights
       real wts(nl,maxna,0:maxnam)
     < fit_neural_network
     > plot_results
```

This is the description of an arc, a bundle of data that is passed from one module to another.

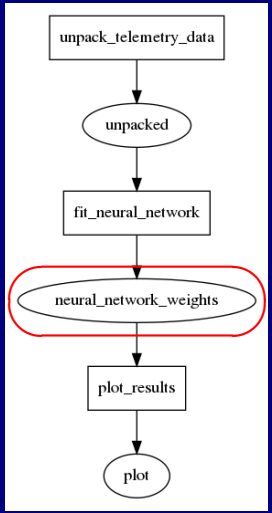
The arcs wrapper language for make



```
\neural_network_weights
        real wts(nl,maxna,0:maxnam)
      < fit_neural_network
      > plot_results
```

- Name of the arc

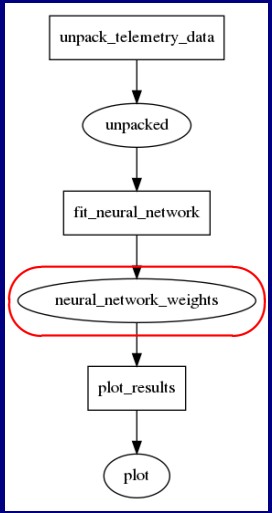
The arcs wrapper language for make




```
\neural_network_weights
        real wts(nl,maxna,0:maxnam)
      < fit_neural_network
      > plot_results
```

- Name of the arc
- Declaration of data content

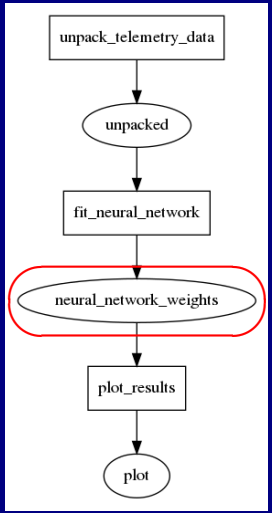
The arcs wrapper language for make




```
\neural_network_weights
      real wts(nl,maxna,0:maxnam)
      < fit_neural_network
      > plot_results
```

- Name of the arc
- Declaration of data content
- Module that outputs the arc

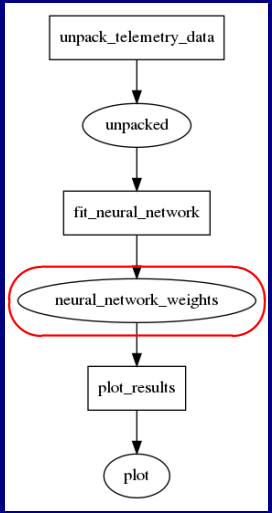
The arcs wrapper language for make




```
\neural_network_weights
      real wts(nl,maxna,0:maxnam)
    < fit_neural_network
    > plot_results
```

- Name of the arc
- Declaration of data content
- Module that outputs the arc
- Module that inputs the arc

The arcs wrapper language for make

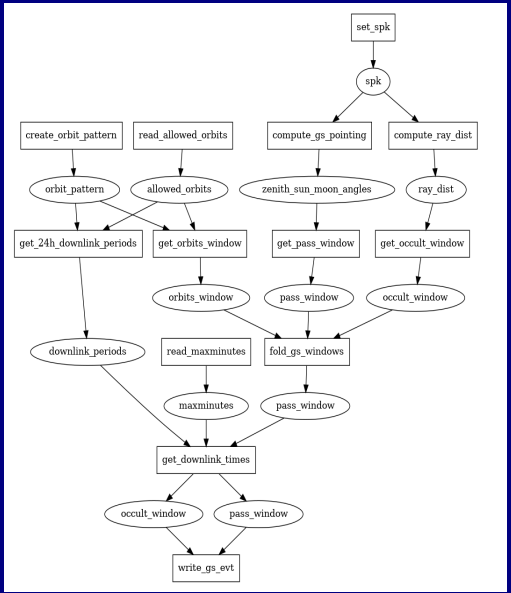



```
\neural_network_weights
      real wts(nl,maxna,0:maxnam)
     < fit_neural_network
     > plot_results
```

- Name of the arc
- Declaration of data content
- Module that outputs the arc
- Module that inputs the arc

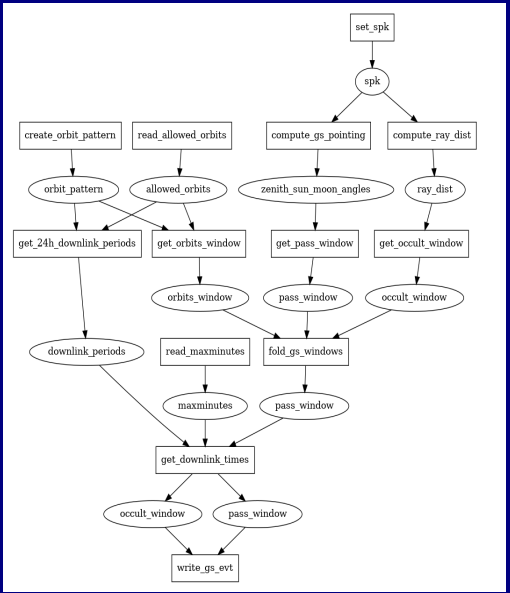
This is the *only* construct of the arcs language.

Science operations reference scenario for EnVision



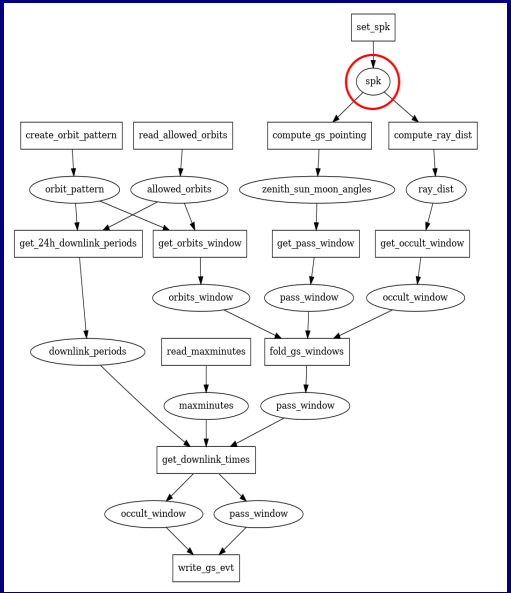
- Envisionary employs functional programming. The main control script is actually a data flow description using the arcs language.
- As such, it makes use of *lazy evaluation*, i. e., only program modules which need to are (re-)executed.
- Modules exchange data via the hard disk, so intermediate results are preserved between program runs.

Data flow description



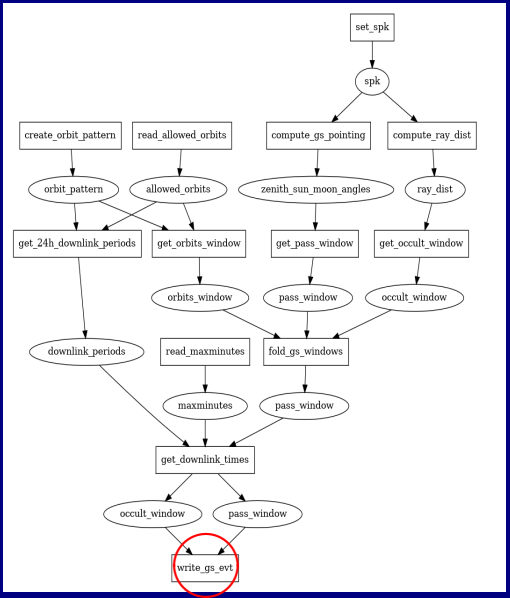
- Such diagrams are automatically created from source code.
- This one displays only a subset (13 out of ≈ 220) of modules to illustrate the principal data flow for the computation of ground station events.
- Boxes are program modules, (ovals) are data structures.

Data flow description



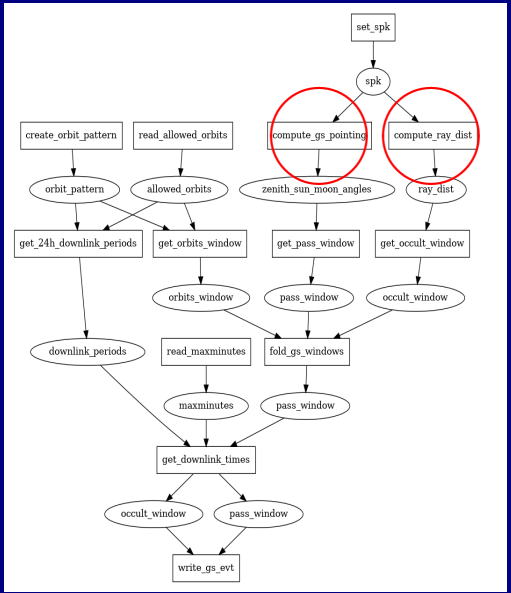
- Such diagrams are automatically created from source code.
- This one displays only a subset (13 out of ≈ 220) of modules to illustrate the principal data flow for the computation of ground station events.
- Boxes are program modules, (ovals) are data structures.
- spk is the SPICE Spacecraft Position Kernel, which contains the spacecraft trajectory.

Data flow description



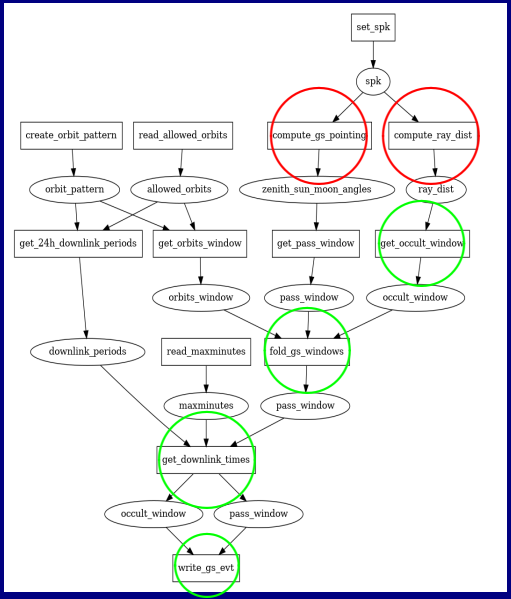
- Such diagrams are automatically created from source code.
- This one displays only a subset (13 out of ≈ 220) of modules to illustrate the principal data flow for the computation of ground station events.
- Boxes are program modules, (ovals) are data structures.
- spk is the SPICE Spacecraft Position Kernel, which contains the spacecraft trajectory.
- write_gs_events is the module which writes the event file for MAPPS.

Lazy evaluation



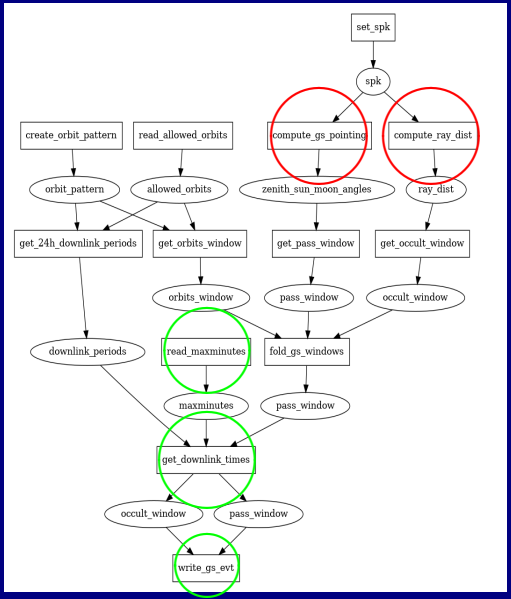
- The two compute_∗ modules are quite time consuming, O(h), everything else takes the wink of an eye.
- We have to re-run these modules (only) if the SPK changes (the spacecraft trajectory).

Lazy evaluation



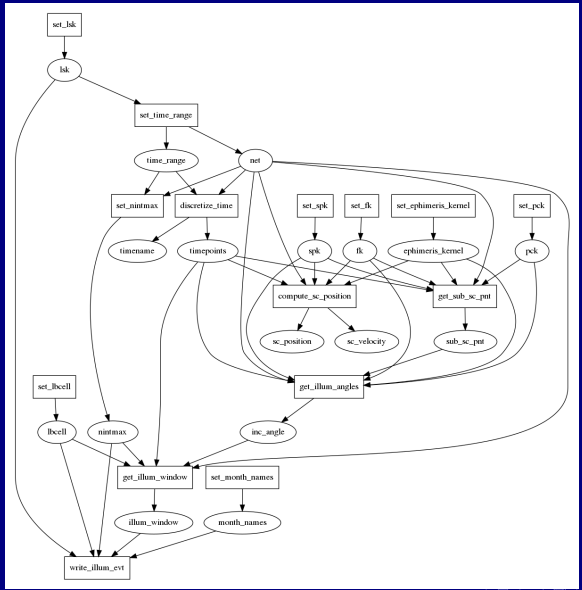
- The two compute_∗ modules are quite time consuming, O(h), everything else takes the wink of an eye.
- We have to re-run these modules (only) if the SPK changes (the spacecraft trajectory).
- If we change the occulting thickness of the atmosphere, only downstream modules are re-executed.

Lazy evaluation

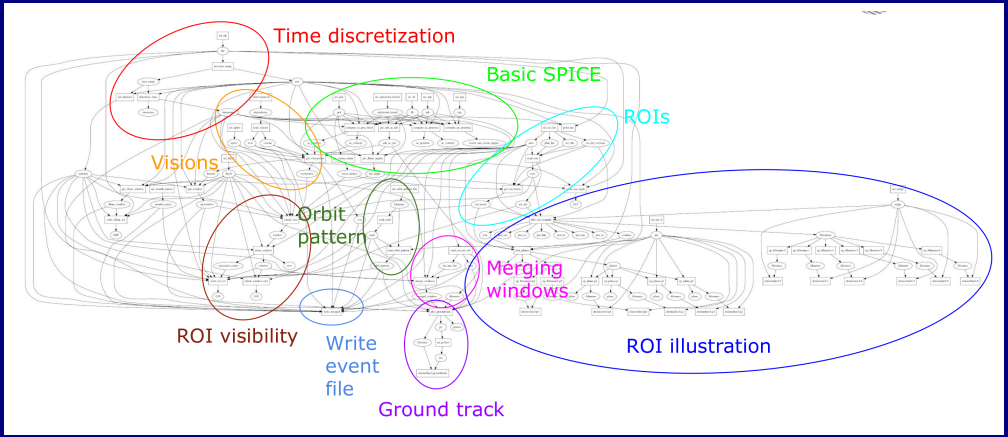


- The two compute_∗ modules are quite time consuming, O(h), everything else takes the wink of an eye.
- We have to re-run these modules (only) if the SPK changes (the spacecraft trajectory).
- If we change the occulting thickness of the atmosphere, only downstream modules are re-executed.
- If the table with daily target downlink times is changed, only downstream modules from *there* are re-executed.

Full Envisionary as of 2019-10-18

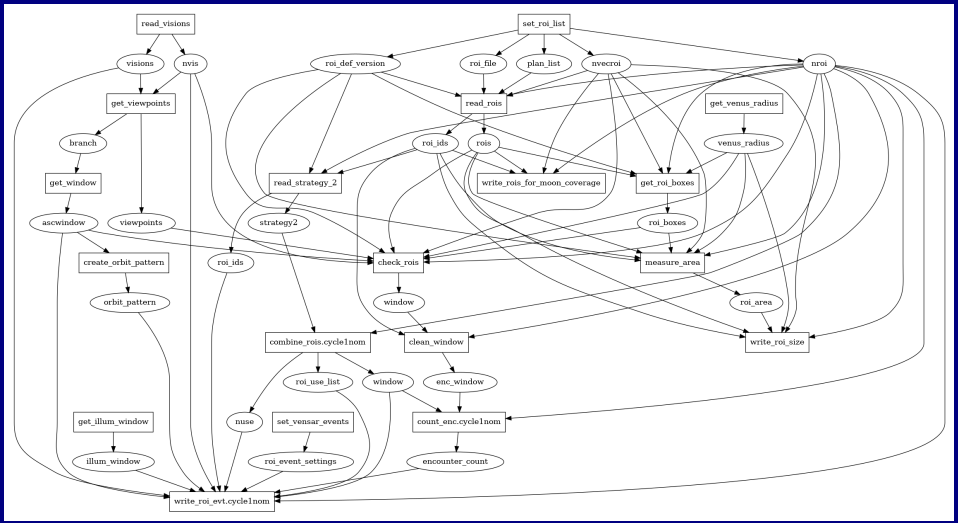


Full Envisionary as of 2020-02-05



By now, Envisionary has grown much too big to be completely displayed in one graph. Selected subsets of modules can be displayed.

Selected modules for ROI coverage estimation



Real “agility”

- No high level plan is needed to get started. Just do first things first.
- The data flow approach is to some extent self documenting.
- The system is runnable at all times.
- Because of lazy evaluation, testing is conveniently done “on the job”.

Quo vadis, Envisionary?

- I shall be retired long before EnVision flies.
- The arcs language is fairly simple and to some extend self-documenting, so anybody could take over the maintenance of Envisionary, but...
- the arcs compiler itself is horrendous spaghetti code, so any needed bug correction or functionality extension would cause the uninitiated bad headache.
- It would be great (but most probably impossible) if the arcs compiler could be properly rewritten to support arbitrary programming languages via configuration files (while it is currently hard wired to support only Fortran, Perl, and OpenDX).

Quo vadis, Envisionary?

- Envisionary employing arcs has proven to be very efficient for the rapid development of a Science Operations Reference scenario.
- At some point, we have to switch (or gradually move) to something else.

Embedding it in a Jupyter Notebook (1)

Cross validation of MAPPS coverage

Besides documenting the work on this issue, this also serves as example of the integration of Envisionary and Moon Coverage in a Jupyter notebook.

Jira issue

ENVISOCMNGT-3

Set up Moon Coverage

Import packages

In [4]:
```
import matplotlib.pyplot as plt

from moon_coverage import TourConfig, MetaKernel, VENUS
from moon_coverage.esa import EsaCremasCollection
from moon_coverage.ticks import date_ticks, km_ticks
from moon_coverage import ROI

from IPython.display import Image
import numpy as np
import pandas as pd
```

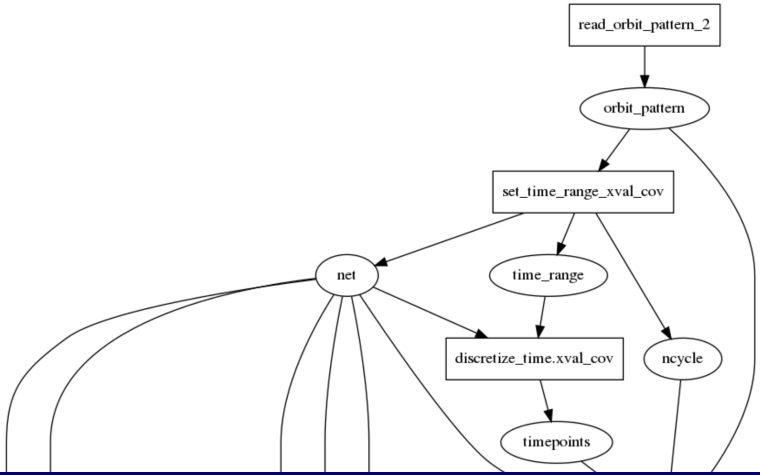
Embedding it in a Jupyter Notebook (2)

Envisionary

Data flow for selected modules

In [5]:
```
Image('../dat/selection_xval_cov.png')
```

Out[5]:



Embedding it in a Jupyter Notebook (3)

Configuration file

In [11]:
```
%%writefile xval_cov.acfg
"""
arcs -d -m compute_coverage,write_grid arcs.f

########################################################################

set_time_range_xval_cov.f
    det = 300

set_npntgc.f
    npntgc = 36
"""
```

```
Overwriting xval_cov.acfg
```

Embedding it in a Jupyter Notebook (4)

Run Envisionary

In [12]:
```
!./envy xval_cov.acfg
```

```
Configuration file: ../exe/xval_cov.acfg
arcs arguments: -d -m compute_coverage,write_grid arcs.f
Configuring ../src/set_time_range_xval_cov.f ...
   Found   det = 300
   OK
Configuring ../src/set_npntgc.f ...
   Found   npntgc = 72
   Setting npntgc = 36
... configuration done.
Starting computation.
make[1]: Entering directory '/lhome/bgrieger/VENUS/ENVISION/SPICE/ENVISIONARY/tmp'
gfortran -O2 -o set_npntgc.x set_npntgc.f
./set_npntgc.x
echo done > set_npntgc
./compute_coverage.x
echo done > compute_coverage
./write_grid.x
echo done > write_grid
make[1]: Leaving directory '/lhome/bgrieger/VENUS/ENVISION/SPICE/ENVISIONARY/tmp'
```

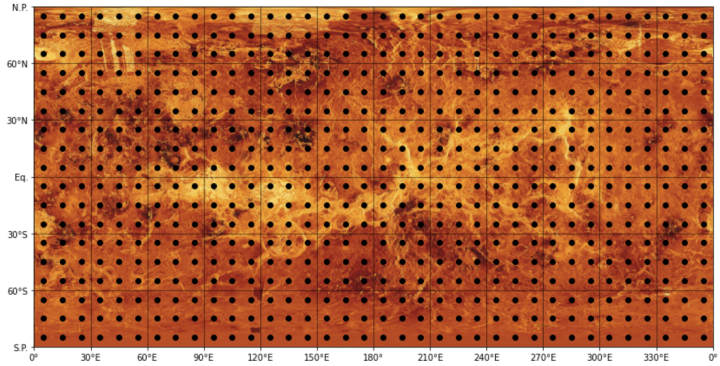
Embedding it in a Jupyter Notebook (5)

Visualize coverage grid

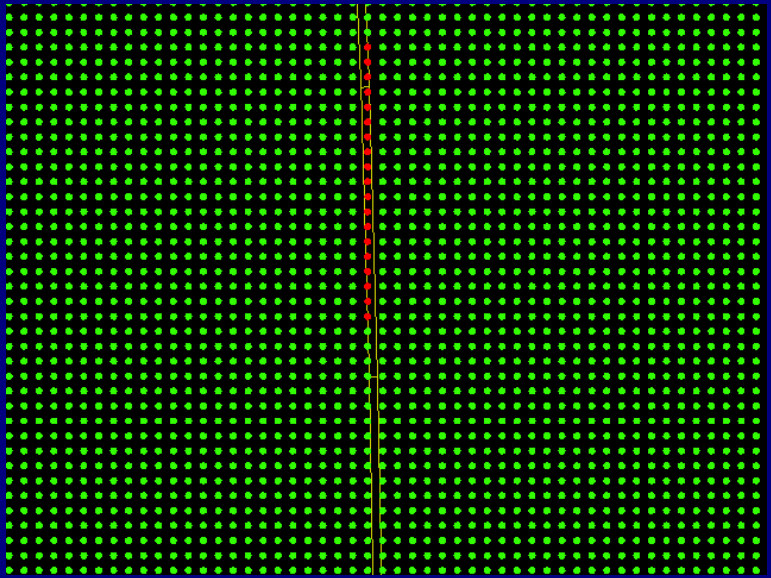
We read the grid coordinates written by Envisionary and plot them on top of a Venus map:

In [13]:
```
grid = pd.read_csv( '../tmp/cov_grid.txt' )
fig = plt.figure(figsize=(14, 9))
ax = fig.add_subplot(projection=VENUS)
ax.plot( grid.Longitude, grid.Latitude, 'o', color='black')
```

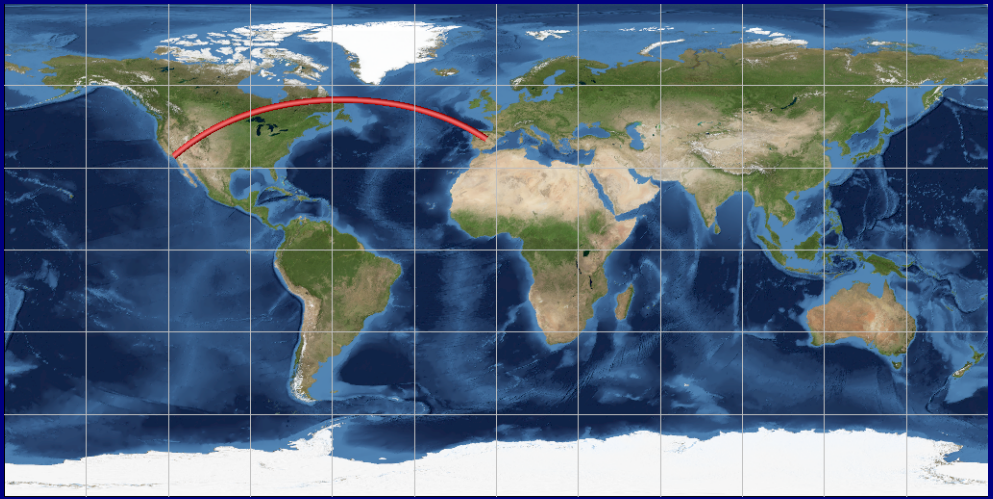
Out[13]:
```
[<matplotlib.lines.Line2D at 0x7f70a97bfb00>]
```



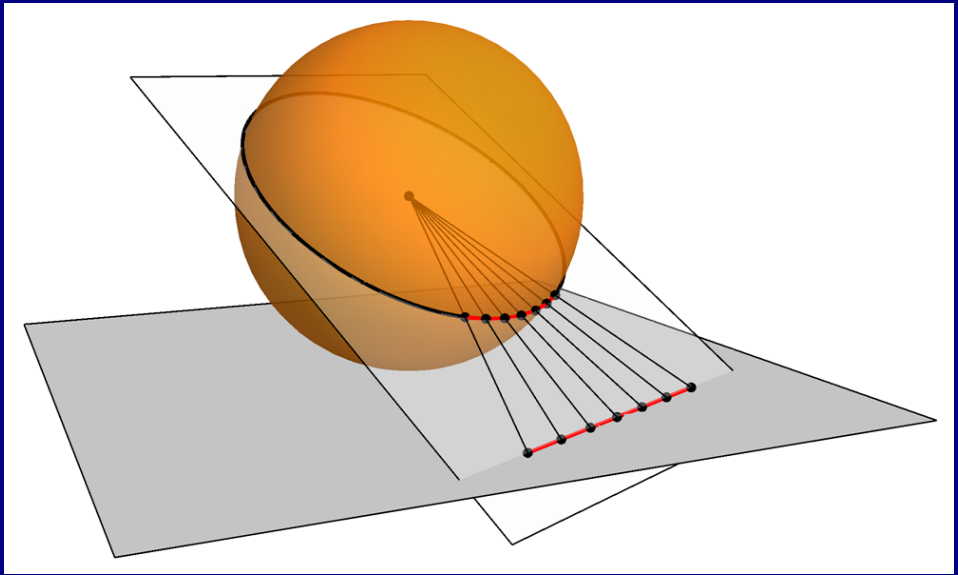
Equidistant cylindrical projection near the equator



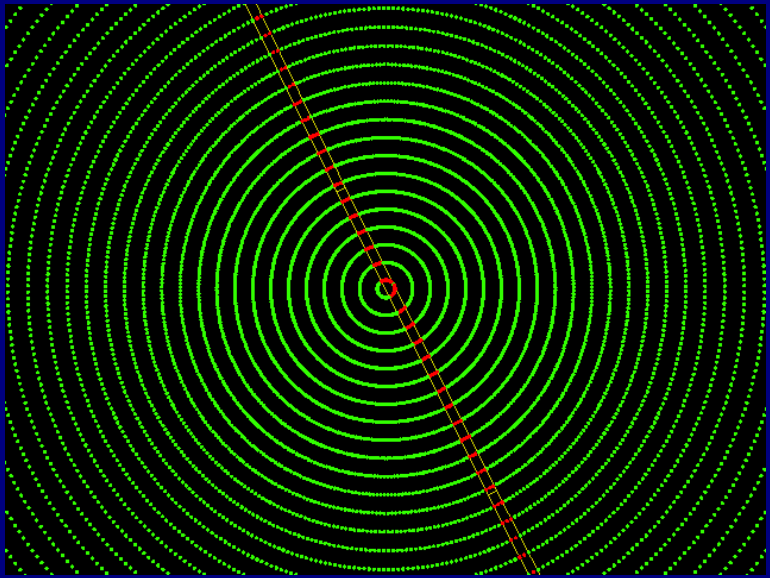
Straight lines are not straight lines!



Gnomonic projection

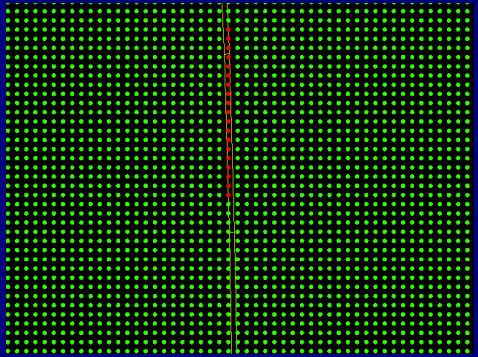


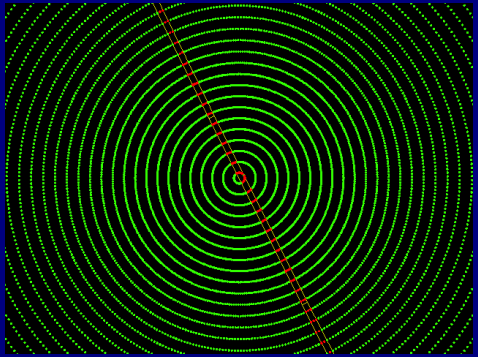
Gnonomic projection near the poles



Three cases

- If a swath crosses the equator, use equidistant cylindrical.
- If a swath lies completely in one hemisphere, use Gomonic for that hemisphere.

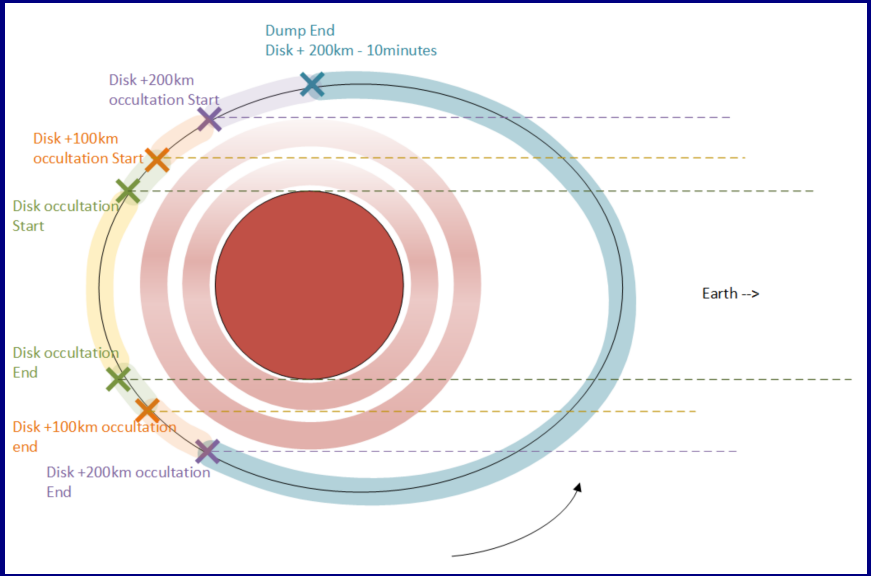




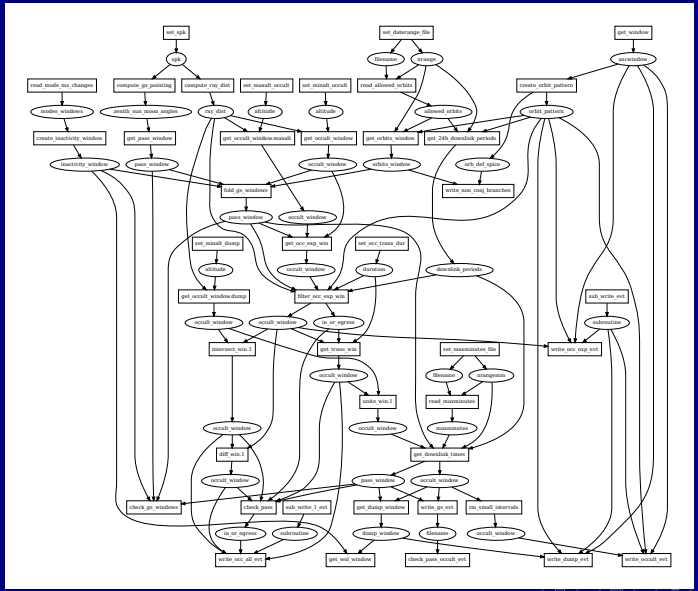
Radio gaga

1. The radio link can either be run in 1-way or 2-way mode.
2. The switching from 2-way to 1-way takes ten minutes. During switching, no radio link is available. The reverse switching from 1-way to 2-way can be considered instantaneous.
3. Dump can be done in either mode, 1-way or 2-way.
4. Gravity can only be done in 2-way mode.
5. Occultation can only be done in 1-way mode.
6. Dump and gravity can only be done above 100 km height (of ground station to spacecraft line of sight).
7. Occultation is done between 0 and 200 km height, with 100–200 km used for calibration.
8. Occultation can only be done at egress if it was done at the preceeding ingress.
9. Target is two occultation ingress/egress pairs per day.

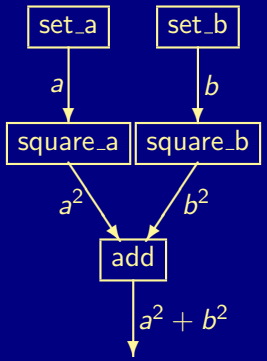
Required events



Mayhem orchestration

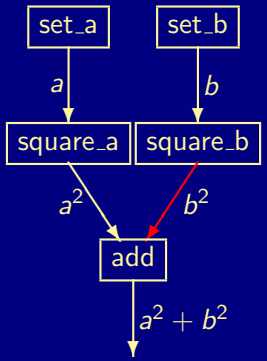


The dataflow C++ template library



- With the arcs language discusssed in the last section, modules are individual programs which exchange data over the hard disk.
- For usual programming with all happening in the RAM, e. g., the language Haskell offers functional programming, using functions to describe a data flow.
- In order to describe directly the dataflow, I wrote a C++ template library. It is object oriented, each module is an object.

Data flow syntax

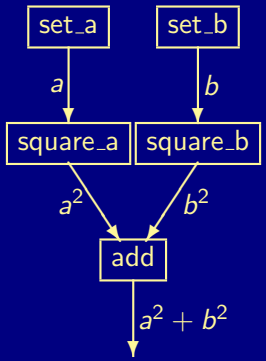


To connect output 0 of module square_b with input 1 of module add, one simply writes in the code:

```
CONNECT( square_b, 0, add, 1 );
```

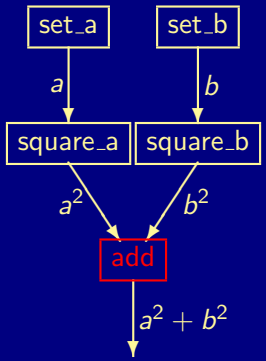
(which is expanded by the C preprocessor to something a little more cryptic, but that you never see)

Flowing up and down



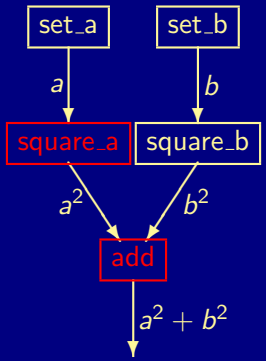
The module add is an object with a public function add.out0() that returns its output 0.

Flowing up and down



The module add is an object with a public function add.out0() that returns its output 0. If this function is called, it calls its own object's protected function update(),

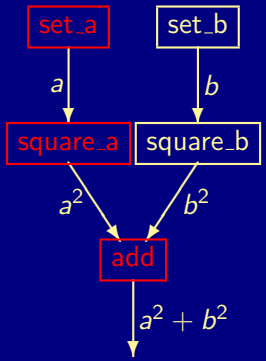
Flowing up and down



The module add is an object with a public function add.out0() that returns its output 0. If this function is called, it calls its own object's protected function update(), and then:

- update() calls the public function square_a.version().

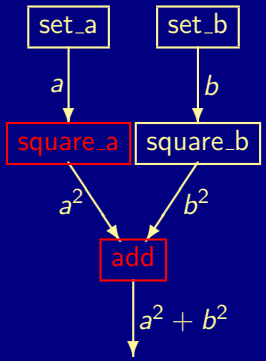
Flowing up and down



The module add is an object with a public function add.out0() that returns its output 0. If this function is called, it calls its own object's protected function update(), and then:

- update() calls the public function square_a.version().
  - square_a.version() calls its own protected function update()
    - ...

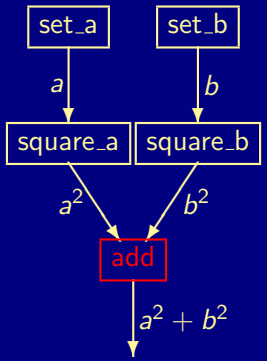
Flowing up and down



The module add is an object with a public function add.out0() that returns its output 0. If this function is called, it calls its own object's protected function update(), and then:

- update() calls the public function square_a.version().
  - square_a.version() calls its own protected function update()
    - ...
  - square_a.version() returns the version of its output.

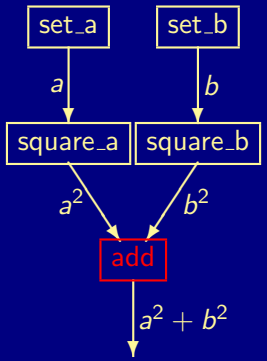
Flowing up and down



The module add is an object with a public function add.out0() that returns its output 0. If this function is called, it calls its own object's protected function update(), and then:

- update() calls the public function square_a.version().
  - square_a.version() calls its own protected function update()
    - ...
  - square_a.version() returns the version of its output.
- If the returned version is newer than the one from the latest computation, update() calls its own protected function compute() and increments the output version.

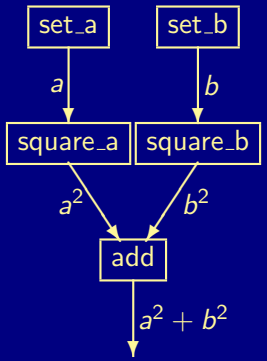
Flowing up and down



The module add is an object with a public function add.out0() that returns its output 0. If this function is called, it calls its own object's protected function update(), and then:

- update() calls the public function square_a.version().
  - square_a.version() calls its own protected function update()
    - ...
  - square_a.version() returns the version of its output.
- If the returned version is newer than the one from the latest computation, update() calls its own protected function compute() and increments the output version.
  - compute() computes the output(s) from the input(s). This is the only user provided function.

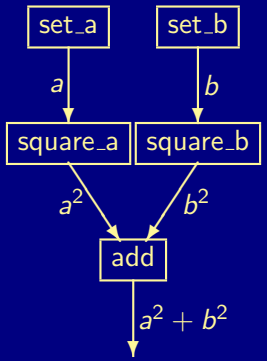
The module object



Each module object has:

- Variables (all protected)
  - Pointers to its input modules
  - Pointers to the outputs of its input modules
  - An array of the versions of its inputs
  - Its outputs
  - Its outputs' version
- Protected functions
  - update(), checks versions and calls compute() *only* if needed.
  - compute(), computes outputs, provided by the user
- Public functions
  - version(), calls update() and returns version
  - instantiation
  - establishing connections (by setting pointers)
  - requesting outputs
  - set() outputs (only inputless modules)

Data flow work flow



All this looks fairly complicated, but it's mostly transparent. To define a data flow, the user only has to

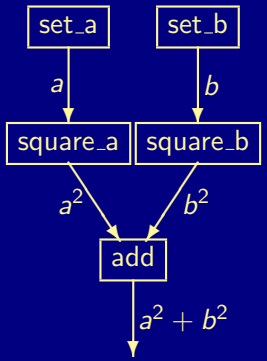
- define module classes by adding compute() functions to templates,
- instantiate module objects (like any object),
- connect module inputs and outputs (with the simple command shown before).

Then the user can

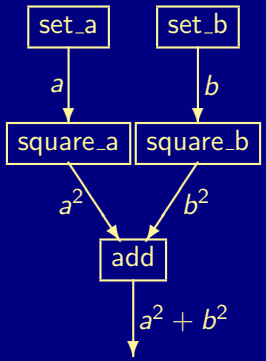
- set inputless modules,
- request module outputs.

On an output request, all necessary computations (and only those) will be performed.

Demo




```
CONNECT( set_a, 0, square_a, 0 );
CONNECT( set_b, 0, square_b, 0 );
CONNECT( square_a, 0, add, 0 );
CONNECT( square_b, 0, add, 1 );
```




```
CONNECT( set_a, 0, square_a, 0 );
CONNECT( set_b, 0, square_b, 0 );
CONNECT( square_a, 0, add, 0 );
CONNECT( square_b, 0, add, 1 );
```

Note to self:

```
cd ~/VULCAN/PROGRAMMING/CPP/CPP_TUTORIAL/src
M df5
```

The dataflow C++ template library

```
template< typename Tin0, typename Tout0 >
class Module_1_1: public Module {
protected:
  Module *m_src0;
  const Tin0 *m_in0;
  int v_in[1];
  Tout0 m_out0;
  void update() {
    int v[1];
    v[0] = m_src0->version();
    bool old = false;
    for ( int i = 0; i < 1; ++i ) {
      if ( v[i] > v_in[i] ) {
        old = true;
        break;
      }
    }
    if ( old ) {
      for ( int i = 0; i < 1; ++i ) {
        v_in[i] = v[i];
      }
      compute();
      ++m_version;
    }
  }
  virtual void compute() = 0;
public:
  Module_1_1() {
    for ( int i = 0; i < 1; ++i ) {
      v_in[i] = 0;
    }
    m_version = 0;
  }
  bool connect0( Module& src, const Tin0 *in ) {
    m_src0 = &src;
    m_in0 = in;
  }
  Tout0 out0() {
    update();
    return m_out0;
  }
  const Tout0 *p_out0() {
    return &m_out0;
  }
};
```

- I implemented this as finger exercise while attending a C++ course.
- It has not yet been applied in the real world, thus I don't know if it is of any real use at all,
- . . . but it was a nice exercise!

Functional programming summary

You may have applied functional programming without knowing it:

- Spreadsheet programs (most probably)
- The ∗nix make utility (maybe)
- OpenDX (maybe not)

Functional programming summary

You may have applied functional programming without knowing it:

- Spreadsheet programs (most probably)
- The ∗nix make utility (maybe)
- OpenDX (maybe not)

Functional programming can be realized without functions by describing directly the data flow.

Functional programming summary

You may have applied functional programming without knowing it:

- Spreadsheet programs (most probably)
- The ∗nix make utility (maybe)
- OpenDX (maybe not)

Functional programming can be realized without functions by describing directly the data flow. Home grown examples:

arcs wrapper language for make: orchestrates data exchange between programs over the hard disk, intermediate results are preserved between program runs.

Functional programming summary

You may have applied functional programming without knowing it:

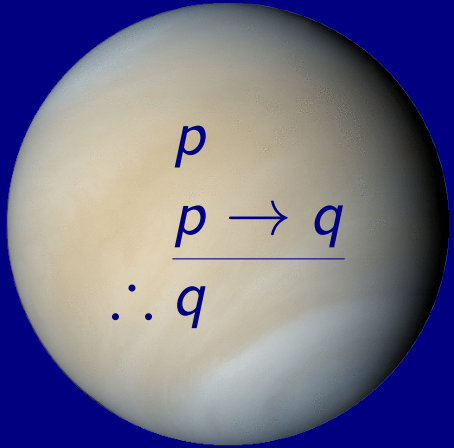
- Spreadsheet programs (most probably)
- The ∗nix make utility (maybe)
- OpenDX (maybe not)

Functional programming can be realized without functions by describing directly the data flow. Home grown examples:

arcs wrapper language for make: orchestrates data exchange between programs over the hard disk, intermediate results are preserved between program runs.

dataflow C++ template library: implements module objects with inputs and outputs that can be connected to define a data flow.

Provision: Employing Prolog in rule based science operations planning



Prolog (programmation en logique) in a nutshell

```
human(socrates).
mortal(X):- human(X).
```

Prolog (programmation en logique) in a nutshell

```
human(socrates).          ⟵ Fact
mortal(X):- human(X).
```

Prolog (programmation en logique) in a nutshell

```
human(socrates).          ⟵ Fact
mortal(X):- human(X).     ⟵ Rule
```

Prolog (programmation en logique) in a nutshell

```
human(socrates).          ⟵ Fact
mortal(X):- human(X).     ⟵ Rule
```

A collection of facts and rules is called a *knowledge base*.

Prolog (programmation en logique) in a nutshell

```
human(socrates).          ⟵ Fact
mortal(X):- human(X).     ⟵ Rule
```

A collection of facts and rules is called a *knowledge base*.

When one or more knowledge bases have been loaded, queries can be posed:

```
?- mortal(socrates).
true.

?- mortal(plato).
false.
```

Inverse query

Extended knowledge base:

```
human(socrates).
human(plato).
mortal(X):- human(X).
```

Inverse query

Extended knowledge base:

```
human(socrates).
human(plato).
mortal(X):- human(X).
```

Direct queries:

```
?- mortal(socrates)
true.

?- mortal(plato).
true.
```

Inverse query

Extended knowledge base:

```
human(socrates).
human(plato).
mortal(X):- human(X).
```

Inverse query:

```
?- mortal(X).
X = socrates ;
X = plato.
```

Provision

- We apply scripts to convert event files written by Envisionary for MAPPS to Prolog knowledge bases.

Knowledge bases

Head of 'roi_knowledge_2.pl':

```
roi_enc_orb_asc(001,3,4275,0).
roi_enc_orb_asc(001,3,4276,0).
roi_enc_orb_asc(001,3,4277,0).
roi_enc_orb_asc(001,3,4278,0).
roi_enc_orb_asc(001,3,4279,0).
```

Head of 'gs_knowledge.pl':

```
gs_pat_orb_asc_cyc(1,10002,0,3).
gs_pat_orb_asc_cyc(1,10017,0,3).
gs_pat_orb_asc_cyc(1,10032,0,3).
gs_pat_orb_asc_cyc(1,10048,0,3).
gs_pat_orb_asc_cyc(1,10063,0,3).
```

Example query

"Tell me, for cycle 1, the orbit branches with orbit pattern number, orbit number, and branch direction, where downlink is geometrically possible and where no ROI observation is planned."

```
:- [roi_knowledge_2].
:- [gs_knowledge].
pass(P,O,A) :- gs_pat_orb_asc_cyc(P,O,A,1),
               \+ roi_enc_orb_asc(_,_,O,A).
run :- forall( pass(P,O,A),
               ( write(P), write('    '),
                 write(O), write('    '),
                 write(A), nl ) ).
```

Query answer

```
1    1320    1
1    381    0
10    190    0
10    2299    1
11    191    0
11    2300    1
12    1177    1
13    1178    1
13    1317    1
13    1517    1
13    3133    1
13    3164    1
13    3210    1
13    3241    1
14    1179    1
14    1318    1
14    1518    1
```

Recursive rules

What is the greatest common divisor (gcd) D of X and Y (with $X, Y, D \in \mathbb{N}$)?

Recursive rules

What is the greatest common divisor (gcd) D of X and Y (with $X, Y, D \in \mathbb{N}$)?

If D is the gcd of X and Y, then D is also the gcd of X and $X + Y$.

Recursive rules

What is the greatest common divisor (gcd) D of X and Y (with $X, Y, D \in \mathbb{N}$)?

If D is the gcd of X and Y, then D is also the gcd of X and $X + Y$.

$$X = n \cdot D$$
$$Y = m \cdot D$$
$$X + Y = (n + m) \cdot D$$

Recursive rules

What is the greatest common divisor (gcd) D of X and Y (with $X, Y, D \in \mathbb{N}$)?

If D is the gcd of X and Y, then D is also the gcd of X and $X + Y$.

$$
\begin{aligned}
X &= n \cdot D \\
Y &= m \cdot D \\
X + Y &= (n + m) \cdot D \\
X + Y &= p \cdot D' \\
X &= q \cdot D' \\
Y &= (p - q) \cdot D'
\end{aligned}
$$

Recursive rules

What is the greatest common divisor (gcd) D of X and Y (with $X, Y, D \in \mathbb{N}$)?

If D is the gcd of X and Y, then D is also the gcd of X and $X + Y$.

Recursive rules

What is the greatest common divisor (gcd) D of X and Y (with $X, Y, D \in \mathbb{N}$)?

If D is the gcd of X and Y, then D is also the gcd of X and $X + Y$.

```
gcd(X,X,X).

gcd(X,Y,D) :- X<Y, Y1 is Y-X, gcd(X,Y1,D).

gcd(X,Y,D) :- Y<X, gcd(Y,X,D).
```

Recursive rules

What is the greatest common divisor (gcd) D of X and Y (with $X, Y, D \in \mathbb{N}$)?

If D is the gcd of X and Y, then D is also the gcd of X and $X + Y$.

```
gcd(X,X,X).

gcd(X,Y,D) :- X<Y, Y1 is Y-X, gcd(X,Y1,D).

gcd(X,Y,D) :- Y<X, gcd(Y,X,D).
```

```
?- gcd(6,4,D).
```

Recursive rules

What is the greatest common divisor (gcd) D of X and Y (with $X, Y, D \in \mathbb{N}$)?

If D is the gcd of X and Y, then D is also the gcd of X and $X + Y$.

```
gcd(X,X,X).

gcd(X,Y,D) :- X<Y, Y1 is Y-X, gcd(X,Y1,D).

gcd(X,Y,D) :- Y<X, gcd(Y,X,D).
```

```
?- gcd(6,4,D).
D = 2
```

Recursive rules

What is the greatest common divisor (gcd) D of X and Y (with $X, Y, D \in \mathbb{N}$)?

If D is the gcd of X and Y, then D is also the gcd of X and $X + Y$.

```
gcd(X,X,X).

gcd(X,Y,D) :- X<Y, Y1 is Y-X, gcd(X,Y1,D).

gcd(X,Y,D) :- Y<X, gcd(Y,X,D).
```

```
?- gcd(10000002,20000001,D).
D = 3
```

Life, Venus and Everything: take-home messages

Life, Venus and Everything: take-home messages

Venus is more beautiful than Mars.

Life, Venus and Everything: take-home messages

Venus is more beautiful than Mars.

Life, Venus and Everything: take-home messages

Venus is more beautiful than Mars.

Life, Venus and Everything: take-home messages

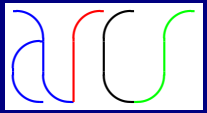
Venus is more beautiful than Mars.

EnVision planning is quite an entertaining puzzle.

Life, Venus and Everything: take-home messages

Venus is more beautiful than Mars.

EnVision planning is quite an entertaining puzzle.



There is programming beyond Python.