The Generic Mapping Tools and Animations for the Masses

P. Wessel[1], F. Esteban[2,3], and G. Delaviel-Anger[4]

[1]Department of Earth Sciences, School of Ocean and Earth Science and Technology, University of Hawaii at Manoa.

[2]Departamento de Ciencias Geológicas, Universidad de Buenos Aires, Facultad de Ciencias Exactas y Naturales, Buenos Aires, Argentina

[3]CONICET-Universidad de Buenos Aires, Instituto de Geociencias Básicas, Aplicadas y Ambientales de Buenos Aires (IGeBA), Buenos Aires, Argentina

[4]Department of Earth and Planetary Sciences, Yale University, New Haven, Connecticut.

Corresponding author: Federico Esteban (esteban@gl.fcen.uba.ar)

Key Points:

- Dynamic content (like animations) can be critical to show changes in science.
- The Generic Mapping Tools software empowers users to easily create animations by taking over non-trivial tasks.
- We encourage scientists to routinely create animations when discussing temporal or spatial changes.

Abstract

The Generic Mapping Tools (GMT) is a well-known set of software originally developed for geosciences, allowing scientists in climate and solid earth disciplines to routinely create publish-ready maps and graphics. However, GMT users rarely make animations despite their undeniable benefit for understanding and teaching dynamical processes. As reading habits shift from print to digital, capitalizing on animations for illustrating scientific concepts is more accessible than ever. In the latest GMT version (6.5) we have added and refined the moving-making modules, alleviating the time-consuming steps that would hinder GMT users from making such animations. In this paper we will explain how GMT ”movie” works then provide six representative examples, from basic to more advanced, to show some of its key features. We hope our presentation will encourage the masses to routinely create animations for their publications.

Plain Language Summary

We live in an era where accessing digital information through all kinds of screens is easily possible. These media allow the display of animations and movies. However, in scientific publications, these formats are scarcely used despite their compatibility with most systems and their clear benefit to convey dynamic concepts. In this paper, we show how The Generic Mapping Tools software facilitates the creation of animations, ranging from simple to complex. With this article, we wish to help and encourage all those involved in scientific outreach to produce pedagogical content that is in tune with the uses of our time.

1 Introduction

The Generic Mapping Tools (GMT; www.generic-mapping-tools.org) is a well-known set of software for the geosciences (Wessel et al., 2013, 2019; Wessel & Smith, 1991, 1995, 1998), in particular in climate and solid earth disciplines. GMT is also a prerequisite for many other well-known software infrastructures, such as MBARI/LDEO’s MB-System (Caress & Chayes, 1995) for multibeam processing and mapping of the seafloor and Scripps Institution of Oceanography’s GMTSAR (Sandwell et al., 2011) for radar interferometric analysis and imaging of crustal deformation. With many tens of thousands of users all over the world it remains essential for data processing, mapping, and plotting workflows.

Scientists routinely make illustrations with GMT but rarely animations, yet understanding change is critical in natural sciences. Anecdotal evidence suggests transformative discoveries were greatly assisted by animations. The transform fault concept in plate tectonics (Wilson, 1965) was difficult to explain, so Tuzo Wilson made paper flipbooks to show its evolution (Eyles, 2022). Propagating rifts (Hey, 1977) were even more complicated, and many remained unconvinced until Richard Hey and students made a computer animation (Atwater, 1981). The American Geophysical Union (AGU) pioneered the online-only journal *Geochemistry, Geophysics, Geosystems* in 2000 and their Information Technology Committee discussed enabling “dynamic content” in AGU journals. Well over 20 years later, AGU (now via publisher Wiley) publishes 25 Earth-science related journals but only two (*Advances* and *Community Science*) allow dynamic content in the article (the others use supplemental archives). Elsevier, another large publisher of science, has over 2600 journals across all fields but nevertheless offers dynamic content in all the journals themselves. As reading habits shift from print to digital, capitalizing on dynamic figures for illustrating scientific concepts is more accessible than ever.

With existing technology requiring minimal effort for implementation (PDFs support animation since 2008, version 1.7 Adobe Extension Level 3), this article aims to encourage publishers and unions to adapt to the times and preferences of readers. This paper is intended for an audience that already uses (or at the very least know about) GMT for their science and geomatics illustrations and we assume that GMT modern mode syntax (Wessel et al., 2019) is familiar to the reader. GMT empowers users to create animations by taking over non-trivial tasks (loop over frames, select data, change variables, determine visible events, and rasterize images to build the final movie). It works in parallel via the available cores on the user's computer. Remains to the user to focus on the figure's content and to write a "master" script that creates one frame as a function of pre-defined variables, and optionally two additional scripts that create static background and foreground layers, sandwiching all frames. Users specify the resolution (e.g., 4k, HD, custom dimensions), format (animated GIF, MP4, etc.), frame rate, titles and fading, and let GMT automatically run large numbers of jobs in parallel or even split them across multiple computers. Although wrappers to GMT exists from MATLAB (Wessel & Luis, 2017), Python (Uieda & Wessel, 2017), and Julia (Luis & Wessel, 2018), at present movie scripts only support Bash, C shell, and DOS batch scripts languages for efficiency. Since GMT's fundamental graphics model is vector based (*PostScript*), we achieve state of the art quality for text, line effects, and symbols. Herein, we will explain how GMT animation works before highlighting some representative GMT animations, from the basic to more advanced, and demonstrate how simple it is to grasp for any GMT user. When AGU/Wiley will allow science-heavy journals like G^3 to embed movies then we can finally celebrate the arrival of dynamic content.

2 Technical Matters

A movie is made up of individual equally sized images that are chronologically sorted then assembled to a movie format, such as MP4, GIF, WebM or any other. Thus, for the purpose of using GMT, it is important that we understand how the aforementioned steps are handled. Many aspects of the resulting movie are optional, but some relate to every movie. Topics that we are required to be familiar with are:

1. The plot canvas onto which we plot our figures and how we define it.
2. The concept of the frame loop where we advance the time (which can be a synthetic index increment) and how our simple movie script applies relevant changes to the plot so that we do not end up with identical frames.

Optional tasks are:

1. Does the movie have static elements that don't require re-processing at each step (e.g. background and foreground images).
2. Perhaps calculations yielding data sets that will be needed by all frames should be run first.
3. Should there be fading in or out of the animation, and how?
4. Should the animation first start with a title sequence?
5. Do we want to embellish the movie with progress indicators and updating labels?
6. Do we want to add an audio track (such as a narration)?

We will go over these items, starting with the required ones. Below (and summarized in Table 1), we will use **bold-face** letters for GMT options, *italics* for parameters, and ***bold-italics*** for

GMT modules (in lower-case letters). The parameters that the movie will set internally (in UPPER-CASE letters) are thus available to your script. As these change with time, your plot changes since it will use some of those parameters to produce variations.

Table 1. Font and format used when referencing parameters, options, and modules.

Parameter	*Formatting*	*Example of usage*
GMT Default setting	Upper-case bold font	**PROJ_LENGTH_UNIT**
GMT module	Lower-case bold font	**grdimage**
GMT Option	Upper-case bold font	**-C**
GMT variable or script	Lower-case italic font	*dpu*
Movie parameter	Upper-case bold-italic font	***FRAME_NUMBER***
Data	Courier font	`2007-04-09T`

2.1 Your Canvas

First, since we are plotting each frame, and GMT users typically are making a plot of some physical size (e.g., often a paper size, say A4 or US Letter), we need to understand how to determine what our "paper size" is so we can do our composition correctly. We call this paper the canvas (Fig. 1) and it is a setting we control. The canvas setting in the ***movie*** module (**-C**) determines basically two things: The size of your "plot paper" and what resolution (in dots per unit; *dpu*) shall this canvas be converted to a raster image. Unless you specify a custom size you are given a canvas size that is either 24 x 13.5 cm (16:9) or 24 x 18 cm (4:3); both standard movie formats but non-standard paper sizes. If your **PROJ_LENGTH_UNIT** setting is *inch* then the custom canvas sizes are just slightly (1.6%) larger than the corresponding SI sizes (9.6 x 5.4" or 9.6 x 7.2"); this has no effect on the size of the movie frames but allow us to use good sizes that work well with the *dpu* chosen. You should compose your plots using the given canvas size, and ***movie*** will make proper conversions of the canvas to image pixel dimensions. Like for regular GMT plots, it is your responsibility to use **-X** and **-Y** to allow for suitable margins and any positioning of items on the canvas. The gray mapping area in Fig. 1 may of course not be rectangular as it depends on your chosen projection and region.

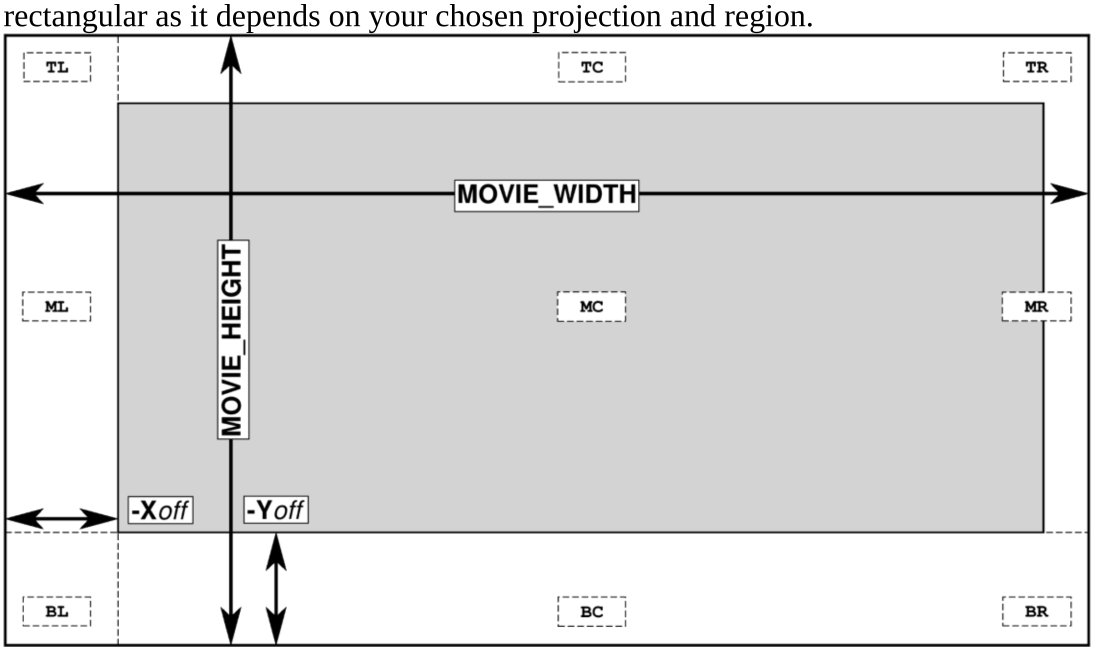

Fig. 1. The ***MOVIE_WIDTH*** and ***MOVIE_HEIGHT*** parameters (Table 2) reflect your canvas dimension. You can use the regular **-X** and **-Y** options to set a logical origin for your intended plot [72p, 72p] and your projection parameters **(-R -J)** indicate the area selected for plotting (gray). The nine dashed boxes show where you may place automatic labels (e.g., frame number, elapsed time, custom text, etc. via the **-L** option) or automatic progress graphs (similarly via the **-P** option).

What about the movie resolution? Well, if you use the standard canvas size then the canvas option only selects the resolution. For instance, if you want to make an HD movie you select **-C**HD. This will compute the required *dpu* so that the rasterized frame is 1920 by 1080 – the standard HD movie pixel size. If you want a 4k movie, then **-C**UHD is your selection. However, if you need a custom canvas (say a portrait movie or a square movie) you will need to tell **-C** both the dimensions and the resolution. For example, selecting a canvas that is 20 cm square and to be rasterized to 600 x 600 pixels (which means a *dpu* of 30) would be **-C**20cx20cx30 (or alternatively **-C**600x600x30c). The ***movie*** command will maintain the canvas dimension and resolution settings as parameters (Table 2) that are available to be used in your shell script instead of hardwiring values that you may forget to change if you try another **-C** setting.

Many animations show some custom indicator of progress. This could be an integer frame number, elapsed real or model time, computed quantities, or values or text read from an input file that defines how many frames there are in the animation (one per data record), to be discussed next.

Table 2. List of constant, variable, and derived ***movie*** parameters.

Constant Movie Parameter	*Purpose or Contents*
MOVIE_NFRAMES	Total number of frames in the movie (via ***movie*** **-T**)
MOVIE_WIDTH	Width of the movie canvas (See Fig. 1)
MOVIE_HEIGHT	Height of the movie canvas (See Fig. 1)
MOVIE_DPU	Dots (pixels) per unit used to convert to image (via ***movie*** **-C**)
MOVIE_RATE	Number of frames displayed per second (via ***movie*** **-D**)
MOVIE_FRAME	Number of current frame being processed
Variable Movie Parameter	*Purpose or Contents*
MOVIE_COL*k*	Numerical value of data column *k* (0, 1, …) for current frame
MOVIE_WORD*k*	Current trailing text split into column *k* (0, 1, …)
MOVIE_TEXT	The entire trailing text for the current frame/record
Derived Movie Parameter	*Purpose or Contents*
MOVIE_ITEM	***MOVIE_FRAME*** but padded with leading zeros
MOVIE_NAME	Movie prefix and underscore before ***MOVIE_ITEM***
MOVIE_FADE	Fading (if set via ***movie*** **-K**) of the current frame (via ***movie*** **-N**)

The derived movie parameters are used for uniquely naming the resulting frame files, with ***MOVIE_NAME*** being the prefix and various suffixes are *.png for the image frames, maybe *.pdf for a test master PDF frame, and perhaps *.gif or *.mp4 for the final movie file. For instance, if you named your movie with **-N**ballon and **-T** implies 1280 frames, then for frame

135 we would have ***MOVIE_FRAME*** being 135, ***MOVIE_ITEM*** expands to 0135 (4 integers are needed for last frame 1279), and ***MOVIE_NAME*** prepends the prefix to yield ballon_0135. While these parameters are available to the *mainscript*, they are mostly used by ***movie*** itself.

2.2 The (Missing) Frame Loop

To build a movie of some length, the ***movie*** module must iterate over all the frames and make slightly different plots (otherwise, the movie would be incredibly boring). Prior to GMT 6.0, ambitious movie makers would have to write complicated scripts where the advancement of frames was explicitly done by a shell loop, and then perhaps that frame counter was used to make some changes to other parameters so that when the plotting started the plot would differ from the previous one. Adding labels and progress indicators were tasks to be scripted up and was quite complicated. At the end of the script, you would have to convert your PostScript plot to a raster image with a name that is lexically increasing, and then later you would use some external software to assemble the movie. Hence, only very brave GMT users attempted to make GMT animations.

GMT 6 (*Wessel et al.*, 2019) changed all that by adding new modules ***movie***, ***events*** and (from 6.1) ***grdinterpolate***. The key idea in ***movie*** is for the user to write a single script that makes one frame, and to obtain change we introduce a series of script variables that will automatically be updated as different frames are built. The loop that was explicit in the scripts for pre-GMT 6.0 movies is gone and instead taken over by ***movie***. So, what are those variables? It depends on how you specify the length of the movie via **-T**. You can simply specify how many frames (e.g., **-T**150 and get frames 0, 1, …, 149), or a specific range of times, such as **-T**120.6/400.6/0.2 for "time" steps of 0.2 from 120.6 to 400.6 or **-T**2021-08-01T/2022-03-01T/1d for 7 months of daily frames. Time can thus be anything that is monotonically increasing through the movie; it is often some form of time but could be distance or similar measures). Very often you have a time-series of some sort whose records correspond to the frames. Hence, the row in the table matches the frame counter (both start at zero). In all these cases, regardless of how time is defined, the parameter ***FRAME_NUMBER*** is always available for use in your script.

While this is useful, animation of a real data set may require access to more variables that change per row (i..e, per frame). As an example, consider these rows from a much larger input file given to ***movie*** **-T**:

```
# time               lon       lat       elevation set  line
2022-01-01T14:00     135.44    33.567    1003.4    A    11
2022-01-01T15:00     135.45    33.567    1107.8    A    12
2022-01-01T16:00     135.46    33.566    1204.9    A    13
```

Our script may need to access these values, but because the loop over frames (i.e., rows) is hidden, we only need a mechanism to access columns. Since all GMT-compliant data tables have [optional] leading numerical columns followed by [optional] trailing text, the numerical entries are named ***MOVIE_COL0***, ***MOVIE_COL1***, etc. and the trailing text is either a single string called ***MOVIE_TEXT*** or we break it into separate words ***MOVIE_WORD0***, ***MOVIE_WORD1***, etc. with option **-T**'s modifier **+w**. Thus, if you need any of those parameters to label the movie via **-L** then they can all be accessed, with customizable formatting.

2.3 Custom Parameters

It can be convenient to define some of your own constant movie parameters so that if you decide to change some of them then anything that derives from those constants will not need to be changed as they will do so automatically. For instance, perhaps you wish to use a variable MY_REGION that specifies the map region and MY_PROJ the projection to be used, then these assignments can be done in a separate script passed to ***movie*** via **-I**. This script will be ingested and added to the hidden machinery that operates under the hood. Thus, you could use your fixed custom variables and compute derived quantities in that script and the derived parameters can then be accessed in your main movie script, just like the ***MOVIE_**** parameters can.

2.4 Static Elements of Animation

To minimize processing time, it is recommended that any static part of the movie be considered either a static background (to be made once by the optional background script; see **-Sb**) and/or a static foreground (to be made once by the optional foreground script; see **-Sf**); ***movie*** will then assemble these layers into each frame automatically. However, instead of a GMT script you could supply a PostScript plot that shall serve as background or foreground instead. The requirement is that the dimensions must match the plot being generated by the *mainscript*. Also, any computation of static data files to be used in the loop over frames can be produced by the background script. Any data or variables that depend on the frame number must be computed or set by the *mainscript* or provided via the parameters as discussed above. Note: Using the variables ***MOVIE_WIDTH*** or ***MOVIE_HIGHT*** to set plot dimensions may lead to clipping against the canvas since these are also the exact canvas dimensions.

2.5 Auto-Generated Labels and Progress Bars

The user's main task is of course to compose the scene and add plot commands to generate the frame plot. A common yet often tedious task is to add either a label that needs updating per frame and some sort of graphic that indicates progress (i.e., how far into the movie are we at this frame). The movie module offers two repeatable options that greatly simplify these tasks. First, let us examine text labels. As shown in Fig. 1, you may place a label anywhere. The nine standard ***text*** justifications can be used as shorthand for placement, but modifiers are available to shift placement relative to these locations. Once you determine where to place the text, the next step is to select what type of text should be placed. The **-L** option offer directives for a static string (no change with frame number – suitable as a fixed title), elapsed time, frame number, percentage of completion, formatting a numerical data column (e.g., ***MOVIE_COL3***), the whole trailing text (***MOVIE_TEXT***) or just a word (e.g., ***MOVIE_WORD2***). A long set of modifiers lets **-L** control fonts, formatting, color, shading, surrounding outlines, and more.

Unlike the label option **-L**, the progress bar option **-P** places one of six progress bars or circles (Fig. 2) at the selected location (same setup as for labels). However, only the three first bars, with directives **a**, **b**, **c**, can be placed anywhere since they are pie or circular arrows. The other three (**d**, **e**, **f**) are linear progress bars of some type and can only be placed on the sides (justifications ML, MR, TC, and TB). As for labels, you can offset these selections to place any progress bar wherever you want. The modifiers to **-P** allows a wide selection of colors and pens and similar adjustments, including a running label. Note that both **-L** and **-P** are repeatable (up to 32 separate items) hence it is simple, for instance, to place both a frame counter in the top left corner and the elapsed time in the top right corner without coding anything into the main movie script.

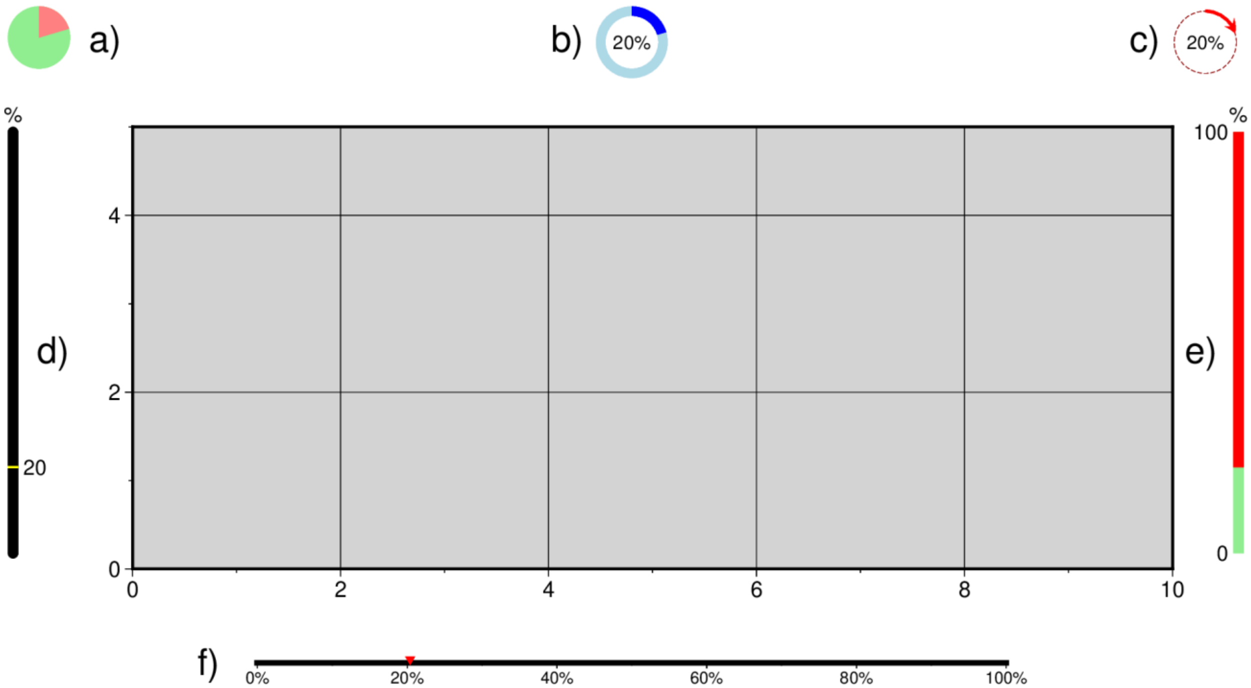

Fig. 2. The six types of movie progress indicators. All have default sizes, placements, colors, and pens (shown) but these can be overridden by the corresponding modifiers to the ***movie*** **-P** option. The linear indicators may only be placed along the side of the canvas while the circular indicators can go in any of the nine reference locations.

2.6 Handling of Event Symbols

A very common feature to be plotted in a movie is what we call an "event". Events have a start and end time and thus a symbol representing such an event should only be visible in the animation while the event is ongoing. Consider you have a large data set of events of which you wish to make an animation. For any given frame (i.e., event time) we need to determine which of the numerous events are active at that point and only plot those, not the others. Checking for this in our movie script would be tremendously tedious and reminiscent of pre-GMT 6 scripting. Because this processing is so involved, we added the module ***events*** to handle this sorting of the data. This module effectuates the events plotting by determining the size of the symbol for an event. It should be zero before and after an event is active, hence we plot the symbol representing the event as long as its size is nonzero. If this is all we do then the animation would, for some frame *N*, suddenly show a new symbol and it would be a constant symbol in size, color, transparency until a later frame *M*, at which point the symbol vanishes because the event has ended. This behavior is illustrated in Fig. 3. Here, the green curves represent the default behavior: Zero size outside the event window. In order to make more interesting movies we have broken the default curve into several optional sub-periods, including ones that begin prior to the event. We call that period the rise time, and by starting the size-amplitude curve for the start of an event we can draw attention to that single event by temporarily boosting its size far beyond its nominal size (here unitary). We then let the symbol remain large during the plateau phase before we let it return to its normal size during the decay phase. Here it stays until the event ends, at which point we may add a fade phase where we shrink the symbol down, possibly to zero. Thus, the resulting solid curve in Fig. 3a illustrates the temporal evolution of a single event symbol as time goes from before the event happens to after the event has ended. This alternative size curve draws attention to the arrival of this event. As Fig. 3 shows, other symbol attributes can also be treated similarly, and separately. Fig. 3b shows that the color of the symbol can be manipulated via an intensity curve (default would be zero), and Fig. 3c shows we can also adjust the transparency of the symbol (Default is no transparency during the event). Another more subtle adjustment is to add a correction to the data sets z-value (e.g., a measure of the event, such as magnitude) which then via the CPT lookup results in a change in the color hue (Fig. 3d).

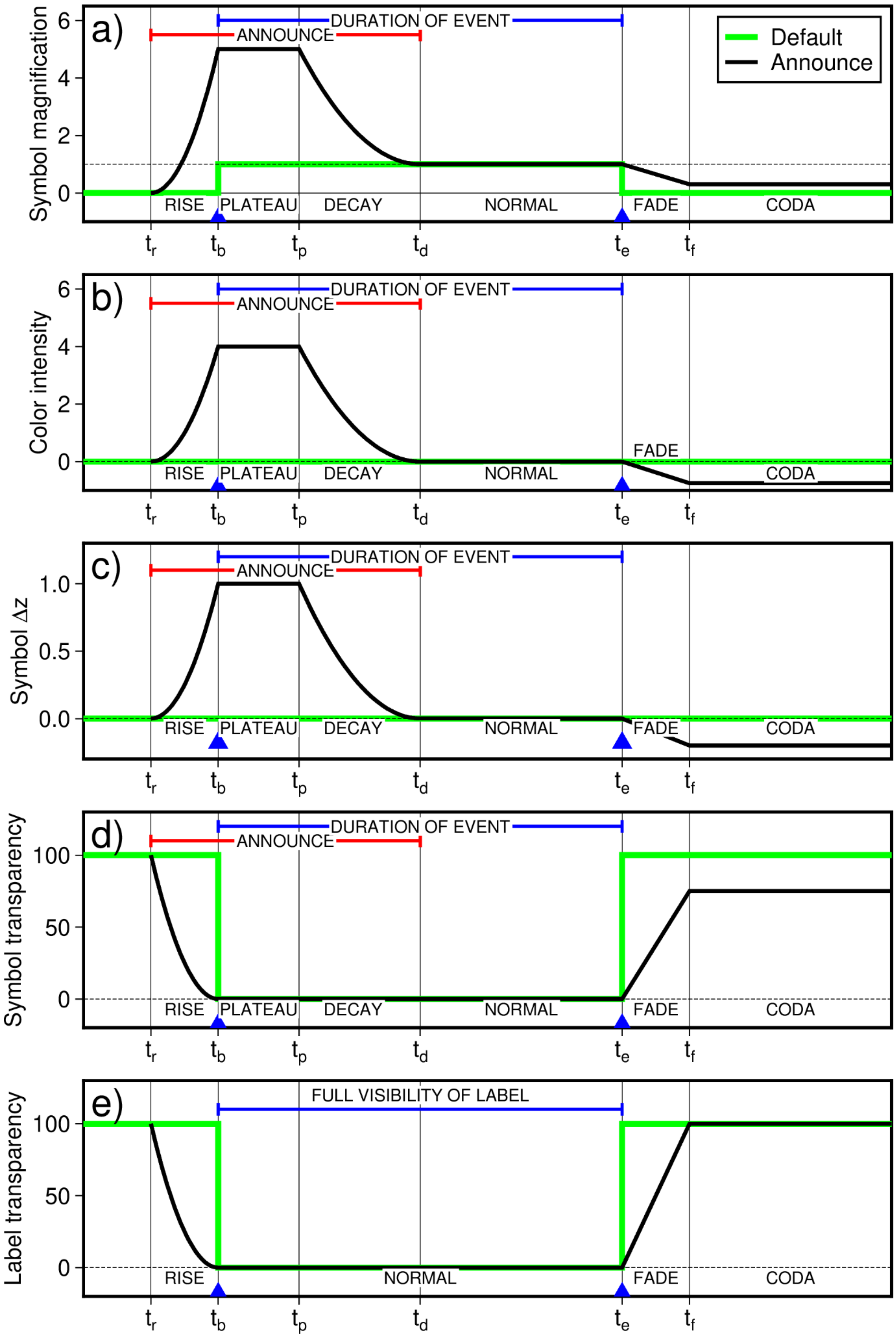
a)
b)
c)
d)
e)
Default
Announce
DURATION OF EVENT
ANNOUNCE
FULL VISIBILITY OF LABEL
RISE
PLATEAU
DECAY
NORMAL
FADE
CODA
Symbol magnification
Color intensity
Symbol Δz
Symbol transparency
Label transparency
t_r
t_b
t_p
t_d
t_e
t_f

Fig. 3. a) Evolution of one symbol's size-variation as a function of time given the time-knots from **-Es** and the magnifications from **-Ms**. Here we temporarily magnify the symbol size five times before decaying to the intended size and eventually shrink it further down to a small size during the fade and coda phases. b) Evolution of one symbol's color intensity as a function of time given the time-knots from **-Es** and the intensities from **-Mi**. Here we seek to whiten the symbol during the event arrival, as well as darken it during a permanent coda phase. c) A symbol may go from being invisible to reaching full opaqueness at the event's beginning time, finally fading to a near-invisible stage after reaching its full duration. e) A symbol may go from a constant color to changing colors at the event's beginning time, finally fading to a different-colored stage after reaching its duration. This curve is scaled by the amplitude [1] and added to the *z*-data before CPT-lookup occurs. e) A label (with or without an accompanying symbol) may go from being invisible to reaching full opaqueness at the event begin time, staying visible for the given duration, and then finally fade away completely. For labels, the plateau and decay periods do not apply.

Events may have labels (if the symbol record has a trailing text). By default, the label is plotted at the same time as the symbol and disappears when the symbol ends. However, we can adjust the transparency curve so that the label appears and disappears more smoothly and even add an offset between the appearance of the symbol and label. With these manipulations one can make a very complicated animation without any scripting beyond playing with ***events's*** **-E** and **-M** options.

In the end, each single event in your data set receives its own (up to five) adjustment curves, all shifted to fit each event's begin and end times, and then ***events*** sorts out which one shall be included in a particular frame and what its attributes will be. We compare the evolution of two circles in Video 1: The green circle uses the default adjustment curves (step-function for size) while the red circle is connected to a more elaborate adjustment curve for size, color intensity, and transparency. We also add labels for both circles and let the green also inherit transparency changes.

2.7 Handling of Lines

Animations of lines (such as ship or satellite tracks) should be able to handle attributes such as variable line thickness and color. However, the PostScript language does not have any operators to handle variable-feature lines. We solve this difficulty in ***events*** by preprocessing line data and converting them to dense point clouds. Now, the **-M** machinery discussed in Section 2.6 can be used to plot the lines via circles whose attributes can change with position and time. Hence, we are able to draw variable-thickness pens and variable-color and -intensity pens since the lines are simply a collection of dense points.

2.8 Interpolation of grids and cubes

Animation of slices through a 3-D volume of evolution of a data set through time (with each time stored in a 3-D cube) may require procession. For instance, the movie requires a constant time-step between frames, yet many data are not sampled equidistantly and hence we will need to interpolate in-between available data slices. We added the new ***grdinterpolate*** module in GMT 6.1 for this reason. This module can handle non-equidistant (in time or depth) cubes or sets of 2-D grids and perform interpolation onto an equidistant array of output times (or depths) required in the movie.

2.9 Title Sequence and Fading

The complete movie may have an optional leading title sequence **(**selected via **-E**) of a given *duration*. A short section at the beginning and/or end of this duration may be designated to fade in/out via the designated fade color [black]. The main animation sequence may also have an optional fade in and/or out section **(-K**). Here, you can choose to fade on top of the animation (which means you lose the first and last frames due to the fading, or you can "freeze" the first and/or last animation frame and only fade over those static images (via modifier **+p**) in order to preserve the whole animation sequence (Fig. 4).

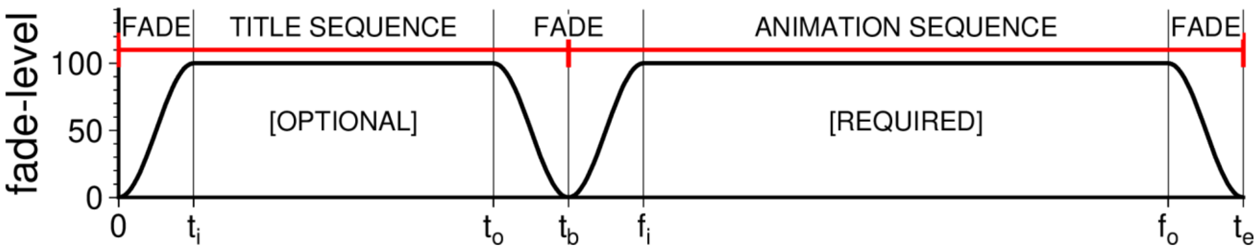

Fig. 5. The complete animation consists of the required animation section, prepended by an optional title sequence. The title sequence can be generated by a script or be given as a PostScript illustration; see ***movie*** **-E**. The fade-levels (0 means black, 100 means normal visibility) can be imposed on the title **(-E**) as well as the animation **(-K**), with many modifiers controlling the effects.

2.10 Adding an audio track

It is possible to add an optional audio track to the complete movie (selected via **-A**). Use **+e** to stretch the audio track to exactly fit the length of the animation (provided the scaling is not less than 0.5 or larger than 2.0). The file is an mp3 format.

3 Examples of GMT Animations

We will show here 6 examples that go from very basic to more complex. We recommend you connect to our public GitHub repository for this paper and try to run the various examples on your own computer [https://github.com/GenericMappingTools/gmt-2024-animation]. At the end of the script execution, a MP4 video and, in most cases, a PNG image will appear in the starting directory with the same name as the script. For this paper GMT version 6.5 should be used (Wessel et al. 2024). Apart from these examples, the GMT YouTube channel (https://www.youtube.com/@TheGenericMappingTools) and the animation gallery (https://docs.generic-mapping-tools.org/dev/animations.html) show many additional movies and the scripts required to make them.

3.1 Spinning Moon

Our first movie is a one-liner, if you don't count the two lines of ***gmt begin*** — ***gmt end*** and the ***movie*** command that runs it all. Given that GMT remote datasets are always on tap from the GMT cloud (and with the GMT 6.5 release we added several celestial bodies such as Mars, Mercury, the Moon, Pluto, and Venus), we decide to image the Moon's topographic relief as we watch it spin 360˚ (Video 2). This tiny 5-line script is all it takes:

```
cat <<- 'EOF' > main.sh
gmt begin
   gmt grdimage @moon_relief_06m -JG-${MOVIE_FRAME}/30/${MOVIE_WIDTH} -Rg -Bg
-X0 -Y0
gmt end show
EOF
gmt movie main.sh -C20cx20cx30 -T360 -Fmp4 -Mf,png -NMovie_Moon
```

The **-T**360 gives frames 0, 1, …, 359 and we use their negatives (i.e., -***$MOVIE_FRAME***) as the central longitude for the azimuthal map (negative so the Moon will spin the right way). Type or copy these < 200 characters into a terminal and a minute later you have a 360-frame MP4 movie of the Moon and a PNG image of the first frame.

3.2 Indiana Jones Map animation

Now we will recreate the iconic flight animation of the Indiana Jones movies. We will show Dr. Jones' flight from New York to Venice, as seen in the "Last Crusade" film (Video 3). Our animation has two scripts: pre.sh and main.sh. In the script pre.sh (which is a background script run just once) we first define the list of stopover cities with their coordinates, which are stored in the cities.txt file and we use ***sample1d*** to interpolate between them every 10 km along the great circle connector and we store the results in distance_vs_frame.txt:

```
cat << 'EOF' > pre.sh
# Dr. Jones stopover cities
cat <<- 'FILE' > cities.txt
-74.007     40.712      New York
-52.712     47.562      St. John's (Newfoundland)
-25.696     37.742      São Miguel (Azores)
-9.135      38.776      Lisbon
 12.342     45.503      Venice
FILE

gmt begin
      gmt sample1d cities.txt -T10k+a > distance_vs_frame.txt
gmt end
EOF
```

Let's analyze this file, which will be later used as input for the **-T** option of the ***movie*** module. It has a total of 767 records, corresponding to the total number of frames of the animation. In table 3 we show the first records, indicating the attribute for each column , and the name of the dynamic variable to be assigned.

Table 3. First records of the distance_vs_frame.txt file.

Longitude ${MOVIE_COL0}	*Latitude* ${MOVIE_COL1}	*Distance from New York (in km)* ${MOVIE_COL2}
-74.007	40.712	0
-73.9053708438	40.7588609404	10.0236735534
-73.8035984365	40.8056326081	20.0473471068
-73.7016826137	40.8523147294	30.071020660
…		

Finally, the critical movie main.sh script will be used to plot the map frames with the flight path. For the first task we use ***coast*** to create a map centered on the plane's changing longitude and latitude (indicated through the dynamic variables, see table 3). The map will have the same width as the canvas as it will not have any offset (due to **-X**0 **-Y**0). For the geographic region, we define the half width (480) and half height (270) in km. Notice that the ratio between them is the same as the canvas of the chosen resolution (16:9), producing a figure area that matches exactly the canvas (i.e. the screen). The other options define the color of the dry and wet areas and the pen for the international boundaries. With the ***events*** module we draw the evolving flight path from the distance_vs_frame.txt file with a thick red line. The distance from New York (indicated through the dynamic variable ${MOVIE_COL2}) will be used to define the "time" of the animation. We use **-Es** for the data to be interpreted as symbols and **-Ar** to indicate that the data should be joined together to form a line or trajectories.

```
cat << 'EOF' > main.sh
gmt begin
   gmt coast -JM${MOVIE_COL0}/${MOVIE_COL1}/${MOVIE_WIDTH} -Y0 -X0 -
R480/270+uk -G200 -Sdodgerblue2 -N1/0.2,-
    gmt events distance_vs_frame.txt -W3p,red -T${MOVIE_COL2} -Es -Ar
gmt end
EOF
```

Finally we use ***movie*** to create the animation from the three previously described files. We made the animation in Full HD resolution (1920 x 1080p) and mp4 format. With -Zs we delete the prefix directory with all the 767 PNG frames and the previously created scripts (pre.sh and main.sh).

```
gmt movie main.sh -Sbpre.sh -N${NAME} -Tdistance_vs_frame.txt -Cfhd -Fmp4 -Zs
-Mf,png -Vi
```

3.2 Indiana Jones Map animation: Complex Version

Now we are going to make a more complex version of the previous animation. The main enhancements are the soundtrack, the title sequence and labels and locations of the cities (Video 4). For the soundtrack, we will use a 35-second trimmed version of "The Raiders March". As we want the whole animation to last as long as the audio, some calculations are necessary. The title sequence will span 6 seconds (to match the intro of the song). The animation sequence itself should last 27 seconds, since we are also going to include transitions at the beginning and at the end of 1 second each. The previous version lasted almost 32 seconds. This was due to the relation between the number of frames (767) and the display rate (24 frames per second by default). That amount of frames was the result of choosing an interpolation interval between cities of 10 km (as we will ignore the fact that our choice implies Dr. Jones' plane travels at a constant speed). Now, for this version we have to make some initial calculations to get the appropriate interpolation interval. First, we include a variable (animation_duration) in in.sh with the time in seconds. Then, in the pre.sh script we do the calculations. We get the total distance to Venice and then the line increment per frame. This increment will depend on the movie rate that we will define in the **movie** command. Here we also add a modifier (**-AR+l**) to get rhumb lines (so they are seen as straight lines called loxodromes in a mercator map).

```
   animation_duration=27 # in seconds
   # Get length of travel and compute line increment in km per frame
   dist_to_Venice=$(gmt mapproject -G+uk cities.txt | gmt convert -El -o2)
   line_increment_per_frame=$(gmt math -Q ${dist_to_Venice} -1
${animation_duration} ${MOVIE_RATE} MUL ADD DIV =) # in km
```

To incorporate the title sequence, we introduce a new script (title.sh), which contains all the necessary information. This includes the title of the animation, accompanying text, and placement of two logos.

```
cat << 'EOF' > title.sh
gmt begin
   echo "12 11.5 Dr. Jones' flight to Venice on his Last Crusade" | gmt text
-R0/24/0/13.5 -Jx1c -F+f26p,Helvetica-Bold+jCB -X0 -Y0
   gmt text -M -F+f14p <<- END
   > 12 6.5 16p 20c j
   We will simulate the flight path from New York to Venice through three
stopovers.
   First, we do some calculations to set a fixed duration of the movie.
   Then, we interpolate between the cities along a rhumb line.
   We also make a separate file for the labels.
   Finally, we make a Mercator map centered on the changing longitude and
latitude.
   We draw the path with a red line. The name of the cities will appear along
with a circle showing its location.
   END
   # Place the GMT logo and Indiana Jones movie logo along the bottom
   gmt image IndianaJones_Logo.png -DjBR+jBR+w0/3c+o2/1c
   gmt logo -DjBL+h3c+o2c/1c
gmt end
EOF
```

To include city names, two commands need to be added. First, in pre.sh we create the label.txt file with the information to determine when to display labels of each city. Then, in main.sh we use **events** to plot the names of the cities and their respective locations.

```
gmt mapproject cities.txt -G+uk > labels.txt
gmt events labels.txt -T${MOVIE_COL2} -L500 -Mt100+c100 -F+f18p+jTC -Dj1c -
E+r100+f100+o-250 -Gred -Sc0.3c
```

Finally, in the ***movie*** command, we use **-A** to add the audio file and we use **-E** to set the properties of the title sequence (6 second duration and a fade out of 1 second). In this case, we create a movie of 60 frames per second (**-D**) and we add a fade in and out, each lasting 1 second for the movie (**-K+p**).

```
gmt movie main.sh -Tdistance_vs_frame.txt -Iin.sh -Sbpre.sh -
Etitle.sh+d6s+fo1s -N${NAME} \
      -AIndianaJones_RaidersMarch.mp3 -Cfhd -Fmp4 -Vi -D60 -K+p -Zs
```

3.4 Messi's goal animation

Let's create a map-based animation showing the progression of Lionel Messi's goals over time until 2023 (Video 5). The animation created will display a main map along with an inset on western Europe and will show a circle of different size and color depending on the amount of goals scored on each game (from 1 to 5). All the information needed is included in the Messi_Goals.txt file (see details in Table 4).

Table 4. Some records of input file Messi_Goals.txt with date, coordinates of the stadium and amount of goals scored in each game.

```
# Game_date   Longitude    Latitude     Goals
2004-01-18    2.118056     41.379722    1
2004-02-08    2.251135     41.457121    1
2004-04-01    2.209621     41.443390    3
…
2023-10-17    -77.033722   -12.067278   2
```

The script can be broken down into five steps. For the first step, we use ***mapproject*** to calculate the height of the main map. It will depend on the region and projection of the main map, and on the width of the canvas. This is important because we are going to use a custom canvas that fits the main map.

```
# 1. Calculate main map/canvas height
main_map_region=-130/145/-40/64 # West/East/South/North boundaries
main_map_projection=W7.5 # Mollweide map center at longitude 7.5
canvas_width=24c
canvas_height=$(gmt mapproject -R${main_map_region} -
J${main_map_projection}/${canvas_width} -Wh)
```

For the second step, we create a file (in.sh) with some variables. These will be used to make the inset map that spans from the continental sectors of Portugal and Spain, to Germany and Great Britain (based on ISO codes and with a small adjustment).

```
# 2. File with variables used for the inset map
cat << 'EOF' > in.sh
# Region, projection, width map and offset in X/Y direction
inset_map_region=PTC,ESC,GB,DE+R1/3/1/-3.5
inset_map_projection=M5.5c # Mercator map of 5.5 cm width
Y=0.2c # Shift plot in in Y-direction
```

```
X=8.5c # Shift plot in in X-direction
EOF
```

In the 3rd step, we create the pre.sh script for two purposes. Firstly, it creates the necessary files for the animation from the input file (Messi_Goals.txt). This involves organizing and scaling the data with ***convert*** (see Table 5).

```
# 1. Reorder and scale data:
gmt convert Messi_Goals.txt -i1,2,3,3+s400,0 > data_scale_by_400.txt
gmt convert Messi_Goals.txt -i1,2,3,3+s80,0 > data_scale_by_80.txt
```

Table 5. Some records of the processed input table.

# Longitude	Latitude	Goals	Goals x400	Date
2.118056	41.379722	1	400	2004-01-18T00:00:00
2.251135	41.457121	1	400	2004-02-08T00:00:00
2.209621	41.44339	3	1200	2004-04-01T00:00:00
…				
-7.033722	-12.067278	2	800	2023-10-17T00:00:00

After computing the cumulative sum of goals over time, we save the data in a file named dates_vs_goals.txt (see Table 6). This file has all the dates every 3 days along with the total number of goals scored up to each respective date. In total it has 2405 records (that will be the amount of frames of the animation).

```
# 2. Create file with dates every 3 days versus cumulative sum of goals
gmt math Messi_Goals.txt -C3 SUM -o0,3 = | gmt sample1d $(gmt info
Messi_Goals.txt -T3d) -Fe -fT > dates_vs_goals.txt
```

Table 6. Some records of file dates_vs_goals.txt.

# Date	Cumulative sum of goals
2004-01-18T00:00:00	1
2004-01-21T00:00:00	1
2004-01-24T00:00:00	1
…	
2023-10-17T00:00:00	885

The second purpose of pre.sh is to make a static background plot. These include both maps (main and inset) and a colorbar. The maps will consist of satellite images with shaded relief and international borders.

```
# 1. Plot main map
# a. Create intensity grid for shadow effect
gmt grdgradient @earth_relief_05m_p -Nt1.2 -A270 -Gmain_intensity.nc -
R${main_map_region}
# b. Plot satellite image with shadow effect and coastlines
```

```
gmt grdimage @earth_day_05m -Imain_intensity.nc -R${main_map_region} \
-J${main_map_projection}/\${MOVIE_WIDTH} -Y0 -X0
gmt coast -N1/thinnest
# c. Create and draw CPT
gmt makecpt \$(gmt info Messi_Goals.txt -T1+c3) -Chot -I -F+c1 -H > Goals.cpt
# Plot colorbar near the bottom left of the canvas with a background panel.
gmt colorbar -CGoals.cpt -DjBL+o0.7c/0.5c+w50% -F+gwhite+p+i+s2p/-2p -L0.1 -
S+y"Goals"
# d. Draw a rectangle showing the area of the inset map
gmt basemap -R\${inset_map_region} -J\${inset_map_projection} -A | gmt plot -
Wthick,white
# e. Plot inset map with zoom in western Europe
gmt inset begin -Dx\${X}/\${Y} -F+p+s -R\${inset_map_region} -
J\${inset_map_projection}
gmt grdgradient @earth_relief_01m_p -Nt1.2 -A270 -Ginset_intensity.nc -
R\${inset_map_region}
gmt grdimage @earth_day -Iinset_intensity.nc
gmt coast -N1/thinnest -Bf --MAP_FRAME_TYPE=plain --MAP_FRAME_PEN=white
gmt inset end
```

In the 4th step we create the main script (main.sh). It uses ***events*** twice to plot the symbols from the files created in the previous step with the same parameters (see Section 2.6 and Fig. 3). We use days as time units. We set the rise phase to 6 (time units) and the decay phase to 18. The symbols are magnified 2.5 times during the rise, and then fade to 0.5 in the coda (Fig. 3A). The color intensity increases 5 times and then returns to its original value (Fig. 3B). The color transparency is set to 0 at the coda (i.e. they are still visible after the duration of the event; see Fig. 3D).

```
# 4. Set up main script
cat << EOF > main.sh
# Set the region, projection and offset (in X and Y) with basemap and then
plot the events.
gmt begin
gmt set TIME_UNIT=d
gmt basemap -R${main_map_region} -J${main_map_projection}/\${MOVIE_WIDTH} -
B+n -Y0 -X0
gmt events data_scale_by_400.txt -SE- -CGoals.cpt -T\${MOVIE_COL0} -Es+r6+d18
-Ms2.5+c0.5 -Mi5+c0 -Mt+c0 -Wfaint
gmt basemap -R\${inset_map_region} -J\${inset_map_projection} -B+n -X\${X} -
Y\${Y}
gmt events data_scale_by_80.txt -SE- -CGoals.cpt -T\${MOVIE_COL0} -Es+r6+d18
-Ms2.5+c0.5 -Mi5+c0 -Mt+c0 -Wfaint
gmt end
```

```
EOF
```

Finally we create the animation with ***movie***. We use the 3 created scripts and the variables defined in step 1 to set the custom canvas dimension. We use ***-L*** to add two labels from the *dates_vs_goals.txt* file (Table 6). The first will be located in the top right of the canvas and will indicate the date which is the first column of the file (**-Lc**0). The second one is in the top left and will show the amount of goals scored at that date. Both labels have a white background panel, border and shadow with the same properties.

```
gmt movie main.sh -Iin.sh -Sbpre.sh -C${canvas_width}cx${canvas_height}cx80 -Tdates_vs_goals.txt \ -N{NAME} -H2 -Ml,png -Vi -Zs -Gblack -Fmp4 -Lc0+jTR+o0.3/0.3+gwhite+h2p/-2p+r \ --FONT_TAG=14p,Courier-Bold,black --FORMAT_CLOCK_MAP=- --FORMAT_DATE_MAP=dd-mm-yyyy \ -Lc1+jTL+o0.3/0.3+gwhite+h2p/-2p+r
```

3.5 3-D density distribution of the Emperor Seamount Chain

In research related to the evolution of hotspot seamount chain, Wessel et al. (2023) developed a fully 3-D evolutionary seamount density model which will be used to load the lithosphere and analyze the flexural deformation taking place below this evolving seamount chain. Video 6 shows moving cross-sections through the chain, allowing one to stop and examine where the densities are higher or lower. We now describe the main commands that allow it (you can see the full script on the GitHub repository). In order to do that we first create a file with along-strike y-profiles (included in the pre.sh script), and then use gmt tools (***select***, ***grdtrack***) to select the data along each profile (in the *mainscript*) and plot them as prisms for the density values, kink lines and topographic outlines.

```
# Select the range of along-strike y-profiles in 2 km increments
gmt math -T-150/150/2 -o1 -I T = pos3D.txt
# Select the densities along the y-profile
gmt select Emperor_oblique_prisms.txt -Z${MOVIE_COL0}/$((${MOVIE_COL0} +
2))+c1 > slice.txt -o0:2,3+o2650,4 -bo3h,2f
 # Plot the densities as columns
gmt plot3d slice.txt -R100/2220/-280/200/0/6000 -Jx0.007c -Jz0.0006c -p165/30
-So1q+b -Crho3D.cpt -X1c -Y0.2c -bi3h,2f
# Draw the kink line
printf "100 ${MOVIE_COL0} 0\n2220 ${MOVIE_COL0} 0\n" | gmt plot3d -W0.25p -p
# Get topography along the profile
gmt grdtrack -GEmperor_oblique_load_mask.nc -
E100/${MOVIE_COL0}/2220/${MOVIE_COL0} -s > topography_profile.txt
# Plot topographic outline
gmt plot3d topography_profile.txt -W0.25p -p -gD0.3c
```

3.6 A Decade of Precipitation

In this last animation (Video 7) we illustrate a time-lapse of daily precipitation around the world. The geographical projection on the left hand side allows us to observe characteristic weather features such as rainforest, cyclones or atmospheric rivers, while the two time-series on the right hand side provide insights on the seasonal cycle and longer trends. The two countries highlighted for this purpose are Argentina (code "AR", south hemisphere) and France (code "FR", north hemisphere) where two of the authors are from. To not further extend the present article with script description, we invite interested readers to review the commented code on GitHub.

4 Advanced Concepts

The limitations with print mode are extensive, and for science to advance we need to be able to make time-variable presentations; this means movies and animations must be acceptable in submitted manuscripts. Currently, with the print mode, it is possible to represent data extending in one dimension (1D) with profiles. It is also possible to plot data (e.g. temperature) over time (or any other dimension, X, Y or Z). It is also possible to use the technique of representing data in 2D where different colors are used to represent the values in 2D. The classic example is a map. Note that in the latter case, the data itself is not displayed but is represented by colors. In these cases the color palette must be chosen carefully to avoid the rainbow palette problem (e.g. Crameri et al., 2020). Another option is 3D blocks. In this case, although 3 dimensions can be visualized, this can only be done from a given perspective. This can sometimes result in data that is not visible or only partially visible.

To avoid the above limitation, it is necessary to display data with one more dimension. Currently there are 2 options to do this: 1. the use of animations (ideal to include the weather). Another alternative is the Universal 3D formats (which include the 3 spatial dimensions). Neither option is available in print format. Some examples of studies would be that of earthquakes and their interrelation as in earthquake swarms. Other examples could be the study of atmospheric sciences (such as temperature) and ocean currents. We illustrated these cases here with animations.

5 Conclusions

It is essential to make it simple to generate temporal variations of models or data and that requires animations or movies. Historically, creating animation has been a guru-level undertaking where export programmers produce grids or columns that vary with time and whose changes are represented by variable geometry or colorization. However, we cannot only let this ability depend on guru experts. We have added moving-making modules to the Generic Mapping Tools to make it simple for any GMT users to make animation. i.e., we desired animations for the masses. Our paper gives an overview of how this can be done and shows many examples (from simple to advanced) to illustrate GMT's capability to make movies. Since most reads of scientific journals now do so in a browser, their tablets, or even phone there are no technical limitations that would prevent illustrations from being animations. We hope our presentation will encourage scientists to routinely create animations when discussing temporal or spatial changes.

Acknowledgements

We thank the US National Science Foundation for their support since 1993. We also acknowledge computational support from Yale Center for Research Computing and the Copernicus Climate Change Service (C3S) at the European Centre for Medium-Range Weather Forecasts (ECMWF) for providing climate data used in the precipitation's animation (Adler et al. 2017). The data used for the Messi animation was compiled by F.E. from the following sources: http://messi.starplayerstats.com/, http://kahkonen.arkku.net/messi and wikipedia.

Open Research Section

The GMT scripts and data used to produce all results and figures and videos presented here are available at https://github.com/GenericMappingTools/gmt-2024-animation under the BSD 3-clause open-source license. Source code and installers for GMT are available at https://www.generic-mapping-tools.org under the GNU LGPL license 3 or later. The releases can also be found at Zenodo (https://zenodo.org/records/10119499).

References

Adler, R., Wang, J. J., Sapiano, M., Huffman, G., Bolvin, D., Nelkin, E., & NOAA CDR Program (2017). Global Precipitation Climatology Project (GPCP) Climate Data Record (CDR), Version 1.3 (Daily) [Global Precipitation at One-Degree Daily Resolution from Multisatellite Observations]. NOAA National Centers for Environmental Information. DOI: 10.7289/V5RX998Z

Atwater, T. (1981). Propagating rifts in seafloor spreading patterns. *Nature*, *290*, 185–186. https://doi.org/10.1038/290185a0

Caress, D. W., & Chayes, D. N. (1995). New software for processing sidescan data from sidescan-capable multibeam sonars. *Proceedings of the IEEE Oceans 95 Conference*, 997–1000.

Crameri, F., Shephard, G. E., & Heron, P. J. (2020). The misuse of colour in science communication. *Nature Communications, 11, 5444*. https://doi.org/10.1038/s41467-020-19160-7

Eyles, N. (2022). *Tuzo: The Unlikely Revolutionary of Plate Tectonics*, Toronto: Aevo University of Toronto Press.

Hey, R. (1977). A new class of "pseudofaults" and their bearing on plate tectonics: A propagating rift model. *Earth and Planetary Science Letters*, *37*, 321–325. https://doi.org/10.1016/0012-821X(77)90177-7

Luis, J. F., & Wessel, P. (2018). A Julia Wrapper for the Generic Mapping Tools, *Eos. Transactions American Geophysical Union*, *Fall Meeting*, (Abstract NS53A-0563).

Sandwell, D. T., Mellors, R., Xiaopeng, T., Wei, M., & Wessel, P. (2011). Open radar interferometry software for mapping surface deformation. *Eos, Transactions American Geophysical Union, 92*(28), 234–235. https://doi.org/10.1029/2011EO280002

Uieda, L., & Wessel, P. (2017). A modern Python interface for the Generic Mapping Tools, Eos. *Transactions American Geophysical Union*, Fall Meeting, (Abstract IN51B-0018).

Wessel, P., & Luis, J. F. (2017). The GMT/MATLAB toolbox. *Geochemistry, Geophysics, Geosystems, 18*, 811–823. https://doi.org/10.1002/ 2016GC006723

Wessel, P., Luis, J. F., Uieda, L., Scharroo, R., Wobbe, F., Smith, W. H. F., & Tian, D. (2019). The Generic Mapping Tools version 6. *Geochemistry, Geophysics, Geosystems*, 20, 5556–5564. https://doi.org/10.1029/2019GC008515

Wessel, P., Luis, J. F., Uieda, L., Scharroo, R., Wobbe, F., Smith, W. H. F., Tian, D., Jones, M., & Esteban, F. (2024). The Generic Mapping Tools version 6.5.0 (6.5.0). Zenodo. https://doi.org/10.5281/zenodo.10119499

Wessel, P., & Smith, W. H. F. (1991). Free software helps map and display data. *Eos, Transactions American Geophysical Union*, *72*(41), 441. https://doi.org/10.1029/90EO00319

Wessel, P., & Smith, W. H. F. (1995). New version of the Generic Mapping Tools released. *Eos, Transactions American Geophysical Union*, *76*(33), 329. https://doi.org/10.1029/95EO00198

Wessel, P., & Smith, W. H. F. (1998). New, improved version of Generic Mapping Tools released. *Eos, Transactions American Geophysical Union*, *79*(47), 579. https://doi.org/10.1029/98EO00426

Wessel, P., Smith, W. H. F., Scharroo, R., Luis, J. F., & Wobbe, F. (2013). Generic Mapping Tools: Improved version released. *Eos, Transactions American Geophysical Union*, *94*(45), 409–410. https://doi.org/10.1002/2013EO450001

Wessel, P., Watts, T., Xu, C., Boston, B., Cilli, P., Dunn, R., & Shilington, D. (2023). Variation in elastic thickness along the emperor seamount Chain. In EGU general assembly 2023, Vienna, Austria, 23–28 Apr 2023, EGU23-6118. https://doi.org/10.5194/egusphere-egu23-6118

Wilson, J. T. (1965). A new class of faults and their bearing on continental drift. *Nature*, *207*, 343–347. https://doi.org/10.1038/207343a0

Video Captions:

Video 1. Two events with the same duration are plotted. The green circle reflects the default adjustment curves while the red follows the more elaborate scaling with a rise, plateau, decay and fading to coda. The labels follow the behavior of the symbols. Red triangle on the time axis indicates present time (***movie* -Pf**) while small red and green circles show variation in symbol size for the two events.

Video 2. Moon spinning. Basic spinning sphere showing the topographic relief of the Moon at 6 min resolution.

Video 3. Simple Indiana Jones map animation. Dr. Jones' flight from New York to Venice, as seen in the “Last Crusade” film.

Video 4. Complex Indiana Jones map animation. This includes a title sequence, audio and the names of the cities.

Video 5. Messi’s goals animation. Location and amount of goals from 2003 until 2023.

Video 6. The 3-D density structure of the Emperor chain. Because the elastic thickness varies along the Emperor it is important to have a well-calibrated density model in order to properly model the deformation and examine variations in slopes.

Video 7. Precipitations around the world. The shading on the embellished global map shows daily-mean precipitation. France (blue) and Argentina (red) are painted to highlight the correspondence with the normalized time-series on the right hand side. A 3-months low-pass filter (boxcar) is also applied to remove high-frequencies signals.