Abstraction Layers in Modern Geospatial Workflows

A choice can be a burden

Chris Reudenbach

2025-12-04

Introduction

Modern geospatial workflows increasingly differ not in *scientific method* but in the **amount of operational complexity they hide**. Whether processing Sentinel-2 imagery or computing terrain derivatives, the core algorithms have not changed. What has changed is how much low-level infrastructure (downloading, tiling, reprojection, cloud masking, indexing, mosaicking, temporary file management) is exposed to the user.

This text explains why **convenience layers** such as `gdalcubes`, CDSE's `GetImage()`, or `Rsagacmd` play a structural role in contemporary geospatial analysis. We compare these abstractions to "pure" R workflows using `terra`, and illustrate the practical impact using two domains:

- **Sentinel-2 processing** (time series, indices, mosaics)
- **SAGA terrain derivatives** (slope, curvature, TPI)

The goal is not to prefer one system over another, but to clarify **what each layer hides, why that matters, and what the trade-offs are**.

The Role of Convenience Layers

A convenience layer is a thin abstraction that removes repetitive, non-scientific tasks. Instead of changing algorithms, it removes the plumbing around them.

Examples of what convenience layers eliminate:

- discovering and filtering STAC assets
- reconstructing tile mosaics and footprints

- handling CRS negotiations
- creating temporary VRTs and memory-safe processing chains
- assembling multi-band stacks
- passing parameters to command-line tools

This is why workflows with abstraction tend to be extremely short (5–15 lines), while **the same scientific operation in pure R often requires 200–400 lines** of bookkeeping.

Convenience layers do *not* alter the scientific logic. They simply prevent infrastructure from overwhelming analysis.

Sentinel-2: Three Processing Paradigms

Sentinel-2 can be processed efficiently in three conceptually distinct ways. The scientific goal—building valid reflectance-based predictors—remains identical; the difference lies in **where the work is performed**.

Comparative Overview (with Performance–Fidelity–Transparency Ratings)

Method	Summary	**Pros**	**Cons**	Perf.	Fid.	Transp.
gdalcubes	Builds a 4D cube (x–y–t–band) and performs lazy mosaicking, reprojection, and temporal aggregation. Excellent for long time series; less suitable for per-date, high-fidelity scenes.	• Extremely fast for multi-year time series • Clean temporal reducers • Memory-safe, minimal code • Automatic mosaics/resampling	• Enforces uniform grid • Loses granule-level detail • Limited cloud-mask control			

Method	Summary	**Pros**	**Cons**	Perf.	Fid.	Transp.
CDSE + GetImage	Executes JS processing scripts on ESA servers (cloud masks, bands, indices). Returns near-native mosaics with ESA QA. Great for high-quality scenes; slower for long time series.	• Near-native 10–20 m resolution • Authentic ESA processing chain • Low disk footprint • High-quality server mosaics	• OAuth required • Slow for long periods • Provider-specific STAC assets • JS scripts must be maintained			
Pure terra workflow	Fully manual local workflow: downloading assets, mosaicking, masking, computing indices. Maximum scientific control; highest code volume; slowest runtime.	• Full scientific control • Transparent and inspectable • Didactically excellent • Reproducible at every detail	• Heavy code volume • Fragile to provider changes • Slow for multi-year AOIs • High maintenance load			

Across the three approaches there is an implicit Pareto balance:

- **gdalcubes** sits near the *performance* corner: very fast and scalable, but with less per-scene fidelity and less step-by-step transparency.
- **CDSE + GetImage** occupies the *fidelity* corner: close to native Sentinel-2 scenes and ESA's own processing, but slower and backend-dependent.
- **pure terra** is closest to the *transparency* corner: every operation is explicit and inspectable, but at the cost of runtime and maintenance effort.

None of them dominates the others across all three axes.
Each method is optimal only **conditional on what you prioritise** (speed, scene fidelity, or methodological transparency).

Applying the Same Logic: The SAGA Example

What happens with Sentinel-2 also did happen with terrain analysis. Computing slope, curvature, TPI, or HAND (Height Above Nearest Drainage) purely in R means you must do everything yourself: fetch DEM tiles, harmonise EPSG codes, mosaic them, crop them, build masks, run low-level operators, and debug all intermediate steps. Almost none of these tasks are geomorphological—they are **administrative overhead**.

`Rsagacmd` eliminates this by wrapping SAGA algorithms in R:

```
saga$ta_morphometry$slope_aspect_curvature(...)
```

The science is identical; the workflow is not. The convenience layer abstracts *every* non-scientific step.

This parallel to Sentinel-2 shows: convenience layers are not shortcuts—they allow researchers to work at the conceptual level rather than the infrastructural one.

Why We Return to STAC Here

All three workflows discussed on this page—**gdalcubes**, **CDSE GetImage**, and **manual terra workflows**—begin with the same fundamental step:
discover Sentinel-2 scenes for the Burgwald AOI.

At first glance this should be straightforward, because all modern EO backends expose Sentinel-2 through **STAC endpoints**.
However, the workflows diverge sharply in behaviour, reproducibility and data structures. Although STAC defines a common schema, it functions much like a library card catalogue: it tells you what exists, but not how the underlying items are stored, formatted, accessed, or processed. STAC standardises the metadata, not the data structures, access mechanisms, or processing behaviour of different providers.

> **STAC = the card index.**
> **Providers = different libraries.**
> **rstac / gdalcubes / CDSE = convenience layers helping you use the index.**

Because the underlying "libraries" differ, the STAC index cannot unify access. Providers expose different asset keys, naming conventions, and access rules, even though the metadata schema is shared.

Examples include:

- different asset names (B04_10m vs B04 vs COG-style filenames)

- different authentication layers (public COGs vs OAuth vs API keys)
- different mosaicking semantics (leastCC, mostRecent, bestPixel)
- different processing levels and QA availability (L1C, L2A, SCL, cloud masks)

As a result, STAC itself is not the abstraction layer — it only indexes what exists. The actual abstraction must be implemented in client libraries such as rstac, gdalcubes, CDSE's GetImage(), or custom code.

> STAC does not enforce canonical labeling. STAC describes *what exists*, not *how* it is generated. A "uniform" STAC search yields highly non-uniform data.

The Cascade of Convenience Wrappers

Because (in our case) the underlying EO backends differ so much, R tooling must build layered convenience wrappers on top.

Layer	Function	Examples
Raw STAC API	Pure catalogue search	`/stac/search`
`rstac`	R interface (paging, fetching)	`stac_search()`, `items_fetch()`
`gdalcubes` STAC adapter	Convert STAC → cube definitions	`stac_image_collection()`
CDSE R package	Provider-specific abstraction	OAuth, JS scripts, ESA mosaics
Your workflow	**Actual geospatial analysis**	**e.g. indices → mosaics → ML**

Each layer abstracts away complexity—but also **introduces provider lock-in**.

This is why:

- `gdalcubes` can use AWS COGs but not CDSE's JS processing
- `CDSE::GetImage` can use ESA's pipelines but not AWS COGs
- `terra` workflows do not bypass STAC; they simply use it at the lowest level—fetching raw asset URLs and handling all processing locally without any higher-level convenience abstraction.

  > STAC is unified. The provider and wrapper logic is not — that's why the workflows diverge.

Choosing the Right Abstraction

Again: There is no "best" method—only one that fits the **balance between control, cognitive load, and performance**.

Use gdalcubes when:

- long time series dominate
- uniform grids are acceptable
- performance is critical

Use CDSE when:

- per-date fidelity matters
- cloud masking and indices should run server-side

Use pure terra when:

- you need exact methodological transparency
- teaching or QA requires inspecting every step

Use generic GIS wrappers like `Rsagacmd`, `qgisprocess`, `rgrass7` when:

- you need high-quality terrain or hydrological derivatives
- you want to reuse the mature, specialised logic of full GIS suites (SAGA, GRASS, QGIS)
- you prefer not to re-implement complex algorithms (TPI, TWI, morphometry, flow models. etc.) in pure R
- you need stable, well-tested operators with decades of domain optimisation behind them
- you rely on `link2GI` to detect, configure, and initialise GIS backends so R can call them transparently

Core Insight

All shown approaches perform the *same science*. They differ only in how much infrastructure they ask the user to handle. The real decision is therefore not "**Which algorithm**?" but **how much low-level responsibility you want to carry**:

- Abstraction reduces boilerplate and shields you from shifting provider APIs.
- Manual control increases transparency but also raises maintenance load, update overhead, and error-susceptibility.
- Proper convenience wrappers minimise this burden by stabilising interfaces and absorbing backend changes.