

Package: sfgpu (via r-universe)

September 27, 2026

Title Explicit CPU, CUDA, and Metal Distances for Projected Points

Version 0.1.0

Description Computes dense Euclidean distance matrices and sparse within-radius point matches for Cartesian two-column coordinates and projected 'sf' POINT geometries. A portable double-precision CPU backend is always available; optional NVIDIA CUDA and Apple Metal backends must be requested explicitly at build time and at each call.

License MIT + file LICENSE

Encoding UTF-8

Suggests sf, units, testthat (>= 3.0.0)

Config/testthat/edition 3

SystemRequirements C++17; optional NVIDIA CUDA toolkit for --enable-cuda; Apple Metal SDK for --enable-metal

RoxygenNote 7.3.3

Config/pak/sysreqs nvidia-cuda-dev

Repository <https://e-kotov.r-universe.dev>

Date/Publication 2026-09-27 15:09:24 UTC

RemoteUrl <https://github.com/e-kotov/sfgpu>

RemoteRef dev

RemoteSha c40de03841be33422ac7628415411468f77336ef

Contents

sfgpu_backends	2
sfgpu_distance	2
sfgpu_install_cuda	4
sfgpu_within_distance	5

Index	7
--------------	----------

sfgpu_backends

Inspect Compiled and Available Backends

Description

Reports whether each backend was compiled into the installed package and whether it is available at runtime. A CPU-only build does not link to GPU runtimes. This function reports capability; it does not select a backend for `sfgpu_distance()`.

Usage

```
sfgpu_backends()
```

Value

A named list with entries `cpu`, `cuda`, and `metal`. Each entry is a list containing logical scalar fields `compiled` and `available`, and character scalar fields `device` and `reason`. The character fields are `NA_character_` when not applicable. `available` reports runtime availability; a GPU build can be compiled but have no usable device.

See Also

`sfgpu_distance`

sfgpu_distance

Compute a Dense Matrix of XY Point Distances

Description

Computes Cartesian Euclidean distances between two coordinate collections. Matrix coordinates use common caller-selected units and carry no CRS interpretation. Spatial inputs must both be spatial objects (either `sf` or `sfc`), with equal known projected CRSs and only nonempty XY point geometries. The function does not transform coordinates or drop extra dimensions. Empty collections are valid. Geographic or unknown CRSs, mismatched CRSs, mixed input kinds, missing or non-finite coordinates, extra dimensions, unsupported geometry types, and empty individual points are errors.

Results are binary64 doubles. CPU and CUDA use double-precision hypot-style arithmetic. Metal uses exact scaled-integer encoding and wide integer distance arithmetic. Pairs that cannot be encoded exactly, approach a rounding boundary, or have a result exponent outside `[-900, 900]` are corrected using CPU hypot. The corrected fraction depends on the input and tile size and can reach 100%. A missing Metal device or command failure still errors. The `max_output_bytes` limit applies to result payload only; tiling bounds execution buffers but does not make the returned matrix out-of-core.

Usage

```
sfgpu_distance(
  x,
  y = x,
  backend = c("cpu", "cuda", "metal"),
  max_output_bytes = 2^53,
  tile_bytes = 64 * 1024^2
)
```

Arguments

<code>x</code>	A numeric two-column matrix of Cartesian coordinates, or an <i>sf</i> / <i>sfc</i> collection of nonempty XY POINT geometries in a known projected CRS.
<code>y</code>	A two-column numeric matrix or spatial input matching the input category of <code>x</code> . Spatial inputs may be <i>sf</i> or <i>sfc</i> and must have the same known projected CRS. Defaults to <code>x</code> .
<code>backend</code>	Backend to use: "cpu", "cuda", or "metal". The default selects CPU. An explicit GPU request errors if the package was built without that backend or no device is available; it never falls back to CPU.
<code>max_output_bytes</code>	Inclusive cap on the dense double result payload in bytes. The default is 2^{53} (unrestricted). This does not cap total process memory.
<code>tile_bytes</code>	Inclusive byte budget for both coordinate input tiles and the output tile. The default is 64 MiB; at least 40 bytes are needed for one pair.

Details

The default installation is CPU-only. CUDA and Metal are optional build features. Use `sfgpu_backends()` to inspect whether each backend is compiled and available at runtime. A GPU-enabled build may still have no usable device.

Value

A numeric matrix with `nrow(x)` rows and `nrow(y)` columns. Input matrix dimnames are propagated. For spatial inputs, the result is a distance matrix carrying units appropriate to the CRS. If `y` is omitted, all pairwise distances within `x` are returned.

See Also

`sf::st_distance`, `sfgpu_backends`

Examples

```
x <- rbind(c(0, 0), c(3, 4))
y <- rbind(c(0, 0), c(5, 4))
sfgpu_distance(x, y)
sfgpu_distance(x)
```

```
if (sfgpu_backends()$cuda$available) {
  sfgpu_distance(x, y, backend = "cuda")
}
```

sfgpu_install_cuda	<i>Install and Enable CUDA Companion DLL on Windows</i>
--------------------	---

Description

On Windows, standard binary builds of sfgpu (such as those from R-universe) are built without NVIDIA CUDA support because standard Rtools uses MinGW GCC, whereas CUDA requires MSVC. `sfgpu_install_cuda()` downloads the pre-compiled companion DLL from GitHub Releases into the user data directory, verifies its SHA-256 checksum against the package manifest, and dynamically loads it.

On Linux, CUDA acceleration is enabled at install time via R CMD INSTALL --configure-args=--enable-cuda .. On macOS, Apple Metal acceleration is compiled automatically on Apple Silicon.

Usage

```
sfgpu_install_cuda(force = FALSE, quiet = FALSE)
```

```
install_cuda(force = FALSE, quiet = FALSE)
```

Arguments

<code>force</code>	Logical; if TRUE, re-download and overwrite any existing DLL. If a CUDA DLL is already loaded in the current R session, R must be restarted first.
<code>quiet</code>	Logical; if TRUE, suppress download progress messages.

Value

Invisibly returns the destination file path of the installed DLL.

See Also

`sfgpu_backends`

sfgpu_within_distance *Find Points Within a Projected Euclidean Radius*

Description

Finds all pairs of points at Euclidean distance less than or equal to the specified radius. Matrix inputs use common caller-selected coordinate units. Spatial inputs must both be `sf` or `sfc`, have the same known projected CRS, and contain only nonempty XY points. The function does not transform coordinates. Empty collections are valid. Geographic or unknown CRSs, mismatched CRSs, mixed input kinds, missing or non-finite coordinates, extra dimensions, unsupported geometry types, and empty individual points are errors. The function returns sparse indices and does not allocate the full pairwise distance matrix. An output cap failure errors without returning a partial result.

Usage

```
sfgpu_within_distance(
  x,
  y = x,
  dist,
  backend = c("cpu", "cuda", "metal"),
  max_output_bytes = 2^53,
  tile_bytes = 64 * 1024^2
)
```

Arguments

<code>x</code>	A numeric two-column matrix of Cartesian coordinates, or an <code>sf/sfc</code> collection of nonempty XY POINT geometries in a known projected CRS.
<code>y</code>	A two-column numeric matrix or spatial input matching the input category of <code>x</code> . Spatial inputs may be <code>sf</code> or <code>sfc</code> and must have the same known projected CRS. Defaults to <code>x</code> .
<code>dist</code>	One finite, non-negative numeric radius. Numeric values use coordinate units for matrices and CRS units for spatial inputs. Spatial inputs also accept one <code>units</code> value convertible to the CRS units.
<code>backend</code>	Backend to use: "cpu", "cuda", or "metal". The default selects CPU. An explicit GPU request errors if that backend was not compiled or no device is available; it never falls back to CPU.
<code>max_output_bytes</code>	Inclusive cap on the output accounting: eight bytes per <code>x</code> list slot plus four bytes per returned index. R object headers and temporary collection workspace are excluded. The default is 2^{53} (unrestricted).
<code>tile_bytes</code>	Inclusive budget for the GPU execution tile, including inputs, output, and scratch. The default is 64 MiB; at least 49 bytes are needed for one candidate pair.

Details

The default installation is CPU-only. CUDA and Metal are optional build features. Use `sfgpu_backends()` to inspect whether each backend is compiled and available at runtime. Distances are inclusive at the radius boundary. A zero radius includes all coordinate duplicates, including the point at the same row position when `y` is omitted.

The output byte accounting includes eight bytes per list slot and four bytes per returned integer index, but excludes R object headers and temporary collection workspace. It is not a cap on total process memory. For spatial inputs, a numeric radius is interpreted in CRS units; a scalar `units` radius is converted to CRS units. For matrix inputs, a `units` radius is invalid.

Membership is decided on the represented binary64 coordinates by the inclusive CPU double-precision predicate `hypot(dx, dy) <= dist`. When the sorted `y` index and bounded query/task/output workspace fit `tile_bytes`, CUDA and Metal sort the points on the host, retain original indices, transfer the index to the device, binary-search `x`-coordinate ranges, filter by `y` coordinate, and compact candidates before the host applies that final predicate. Metal stores ordered binary64 keys. If the index workspace does not fit, either GPU uses its bounded tiled rectangle scan; a 49-byte tile selects this path. CUDA compacts at most 1,048,576 candidates per wave and Metal at most 65,536 per command. The CPU backend sorts and indexes the `y` points by `x` coordinate to prune candidates before applying the same predicate. A tiled GPU scan can examine `nrow(x) * nrow(y)` pairs, though no backend creates the dense distance matrix. CUDA, Metal, and CPU have workload-dependent costs; no speed advantage is guaranteed, and CPU remains the default.

Value

A list with one integer vector per row of `x`. Each vector contains the increasing, one-based indices of rows in `y` whose Euclidean distance is less than or equal to `dist`. Equal-coordinate duplicates are included. The list is named with the row names of `x`, when present.

See Also

`sfgpu_distance`, `sf::st_is_within_distance`, `sfgpu_backends`

Examples

```
x <- rbind(c(0, 0), c(3, 4))
y <- rbind(c(0, 0), c(5, 4))
sfgpu_within_distance(x, y, dist = 2)
sfgpu_within_distance(x, dist = 0)

if (sfgpu_backends()$cuda$available) {
  sfgpu_within_distance(x, y, dist = 5, backend = "cuda")
}
```

Index

`install_cuda(sfgpu_install_cuda)`, 4

`sfgpu_backends`, 2

`sfgpu_distance`, 2

`sfgpu_install_cuda`, 4

`sfgpu_within_distance`, 5