Package 'qvivid'

August 9, 2026

Title Fast Quantum Simulation and Vivid Visualizations

Version 0.1.2

Description Builds, simulates, inspects, and animates gate-based quantum circuits. The package provides a readable reference implementation and compiled state-vector kernels, reproducible shot sampling, circuit diagrams, phase-aware state plots, reduced-state Bloch spheres, journal-sized figure export, and animated trajectories. For background on the implemented methods, see Nielsen and Chuang (2010, ISBN:9781107002173).

License MIT + file LICENSE

URL https://github.com/SanmiAndreSofa/qvivid

BugReports https://github.com/SanmiAndreSofa/qvivid/issues

Encoding UTF-8

RoxygenNote 7.3.2

Depends R (>= 4.2.0)

Imports grDevices, graphics

Suggests gifski, ggplot2 (>= 3.4.0), knitr, ragg (>= 1.2.0), rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

NeedsCompilation yes

Author Sanmi (Oluwasanmi) Adenaiye [aut, cre, cph]

Maintainer Sanmi (Oluwasanmi) Adenaiye <dradenaiyesanmi@gmail.com>

Repository CRAN

Date/Publication 2026-08-09 06:50:02 UTC

Contents

`qvivid-package` *Fast Quantum Simulation and Vivid Visualizations*

Description

Builds, simulates, inspects, and animates gate-based quantum circuits with an R-native API, a compiled statevector backend, reproducible sampling, phase-aware graphics, reduced-state Bloch spheres, and journal-sized exports.

Details

The package uses one-based qubit indices. Qubit 1 is the least significant statevector bit, while displayed basis labels use conventional high-to-low order. Use `quantum_circuit` to begin.

`animate_bloch` *Animate a Bloch Trajectory*

Description

Reduces a selected qubit at every recorded execution step, renders consistent Bloch-sphere frames, and encodes them with the optional `gifski` package.

Usage

```
animate_bloch(
  result,
  file,
  qubit = 1L,
  fps = 3,
  width = 720,
  height = 720,
```

```
  theme = c("nature", "npj", "colorblind", "dark", "light", "mono"),
  view = c("perspective", "xy", "xz", "yz"),
  trail = TRUE,
  loop = TRUE,
  progress = interactive()
)
```

Arguments

result	A qv_result created with record = TRUE.
file	Destination path ending in .gif.
qubit	A one-based qubit index.
fps	Positive frames per second.
width, height	Output dimensions in pixels.
theme	Nature-style, npj-inspired, colorblind, dark, light, or monochrome preset.
view	Perspective sphere or a planar projection.
trail	Retain the path through earlier frames.
loop	Repeat the GIF indefinitely.
progress	Display encoder progress interactively.

Value

A qv_animation containing the normalized GIF path and metadata.

Examples

```
if (requireNamespace("gifski", quietly = TRUE)) {
  local({
    gif_file <- tempfile(fileext = ".gif")
    on.exit(unlink(gif_file), add = TRUE)
    result <- quantum_circuit(1L) |>
      gate_h(1L) |>
      simulate_quantum(backend = "reference", record = TRUE)
    animate_bloch(
      result,
      gif_file,
      width = 320L,
      height = 320L,
      progress = FALSE
    )
  })
}
```

`animate_state` *Animate Quantum State Evolution*

Description

Renders a gate-by-gate probability and phase trajectory and encodes it with the optional `gifski` package.

Usage

```
animate_state(
  result,
  file,
  fps = 3,
  width = 960,
  height = 760,
  top = NULL,
  theme = c("nature", "npj", "colorblind", "dark", "light", "mono"),
  include_circuit = TRUE,
  loop = TRUE,
  progress = interactive()
)

## S3 method for class 'qv_animation'
print(x, ...)
```

Arguments

`result`	A `qv_result` created with `record = TRUE`.
`file`	Destination path ending in `.gif`.
`fps`	Positive frames per second.
`width, height`	Output dimensions in pixels.
`top`	Maximum basis states shown in every frame.
`theme`	Nature-style, npj-inspired, colorblind, dark, light, or monochrome preset.
`include_circuit`	Include a synchronized circuit execution playhead.
`loop`	Repeat the GIF indefinitely.
`progress`	Display encoder progress interactively.
`x`	A `qv_animation`.
`...`	Additional arguments reserved for methods.

Value

A `qv_animation` containing the normalized GIF path and metadata.

Examples

```
if (requireNamespace("gifski", quietly = TRUE)) {
  local({
    gif_file <- tempfile(fileext = ".gif")
    on.exit(unlink(gif_file), add = TRUE)
    result <- quantum_circuit(1L) |>
      gate_h(1L) |>
      simulate_quantum(backend = "reference", record = TRUE)
    animate_state(
      result,
      gif_file,
      width = 240L,
      height = 240L,
      include_circuit = FALSE,
      progress = FALSE
    )
  })
}
```

---

bloch_vector *Bloch Vectors, Spheres, and Trajectories*

Description

Computes a reduced single-qubit state directly from a statevector and renders it as a publication-oriented Bloch sphere. Entanglement and other mixing appear as contraction of the vector inside the unit sphere.

Usage

```
bloch_vector(x, qubit = 1L)

trajectory_bloch(result, qubit = 1L)

plot_bloch(
  x,
  qubit = 1L,
  theme = c("nature", "npj", "colorblind", "dark", "light", "mono"),
  view = c("perspective", "xy", "xz", "yz"),
  trajectory = FALSE,
  main = NULL,
  subtitle = NULL
)

## S3 method for class 'qv_bloch'
plot(x, ...)

## S3 method for class 'qv_bloch'
print(x, ...)
```

Arguments

x	A complex statevector, qv_result, qv_bloch, or qv_bloch_trajectory.
result	A qv_result created with record = TRUE.
qubit	A one-based qubit index.
theme	Nature-style, npj-inspired, colorblind, dark, light, or monochrome preset.
view	Perspective sphere or the xy, xz, or yz planar projection.
trajectory	Draw the complete recorded path for a result.
main, subtitle	Optional title and subtitle.
...	Arguments passed to plot_bloch(); additional print arguments are reserved.

Value

bloch_vector() returns a qv_bloch object. trajectory_bloch() returns a data frame with one vector per step. plot_bloch() returns the plotted vector invisibly.

Examples

```
result <- quantum_circuit(2, name = "Bell state") |>
  gate_h(1) |>
  gate_cx(1, 2) |>
  simulate_quantum(record = TRUE, backend = "reference")

bloch_vector(result, qubit = 1)
plot_bloch(result, qubit = 1, trajectory = TRUE)
```

---

measure *Add Terminal Measurements*

Description

Maps terminal computational-basis measurements from qubits to classical bits. Measurements must be terminal in qvivid 0.1.x.

Usage

```
measure(circuit, qubits, clbits = qubits)

measure_all(circuit)
```

Arguments

circuit	A qv_circuit.
qubits	One-based qubit indices.
clbits	Matching one-based classical-bit indices.

Value

A modified qv_circuit.

Examples

```
circuit <- quantum_circuit(2, n_clbits = 3) |>
  gate_h(1) |>
  measure(qubits = c(1, 2), clbits = c(3, 1))
circuit
```

---

plot_state *Visualize Quantum States and Circuits*

Description

Plots phase-aware probability bars, themed circuit diagrams, and synchronized circuit/state execution views. Probability is encoded by bar height and complex phase by cyclic fill color. The default state-plot behavior does not depend on ggplot2 or any other suggested package being installed.

Usage

```
plot_state(
  x,
  top = NULL,
  theme = c("nature", "npj", "colorblind", "dark", "light", "mono"),
  engine = c("base", "ggplot2", "auto"),
  main = NULL,
  subtitle = NULL
)

plot_circuit(
  circuit,
  highlight = NULL,
  theme = c("nature", "npj", "colorblind", "dark", "light", "mono"),
  main = NULL,
  subtitle = NULL
)

plot_execution(
  result,
  step = NULL,
  top = NULL,
  theme = c("nature", "npj", "colorblind", "dark", "light", "mono"),
  main = NULL,
  subtitle = NULL
)
```

```
theme_quantum(
  theme = c("nature", "npj", "colorblind", "dark", "light", "mono"),
  base_size = NULL
)

qv_palette(
  theme = c("nature", "npj", "colorblind", "dark", "light", "mono")
)

## S3 method for class 'qv_result'
plot(x, ...)

## S3 method for class 'qv_circuit'
plot(x, ...)
```

Arguments

x	A statevector, simulation result, or circuit for the method.
top	Maximum basis states selected by probability.
theme	Nature-style, npj-inspired, colorblind, dark, light, or monochrome preset.
engine	Rendering engine. "base" (the default) draws immediately and returns the plotted data invisibly. "ggplot2" returns a ggplot object for explicit printing or composition. "auto" is retained as a backward-compatible alias for "base" and never changes according to installed optional packages.
main, subtitle	Optional plot title and subtitle.
circuit	A qv_circuit.
result	A qv_result.
step	Optional recorded execution step.
highlight	Optional operation index emphasized in the circuit.
base_size	Optional base ggplot2 text size; the preset supplies a default.
...	Arguments passed from S3 methods.

Value

The ggplot2 state engine returns a ggplot object visibly. Base state plots return a data frame containing their plotted state data invisibly; base circuit plots return their circuit invisibly. Theme and palette functions return their respective objects.

quantum_circuit *Create and Inspect Quantum Circuits*

Description

Creates the validated circuit intermediate representation used by every qvivid backend and visualization.

Usage

```
quantum_circuit(n_qubits, n_clbits = n_qubits, name = NULL)

circuit_depth(circuit)

## S3 method for class 'qv_circuit'
print(x, ...)
```

Arguments

n_qubits	Number of qubits, using one-based R indices.
n_clbits	Number of classical bits.
name	Optional human-readable circuit name.
circuit, x	A qv_circuit.
...	Additional arguments reserved for methods.

Value

quantum_circuit() returns a named list with class qv_circuit. Its stable 0.1.x fields are name, n_qubits, n_clbits, operations, and schema_version. Operation records expose type, name, label, qubits, clbits, matrix, and parameters; treat these fields as read-only. circuit_depth() returns one integer. The print method returns its input invisibly.

Examples

```
circuit <- quantum_circuit(2, name = "Bell state")
circuit
circuit_depth(circuit)
```

`save_quantum_plot` *Export a Publication-Ready Quantum Figure*

Description

Opens a correctly sized vector or high-resolution raster device, draws a qvivid view with a consistent preset, and writes the destination only after rendering has completed successfully.

Usage

```
save_quantum_plot(
  x,
  file,
  view = c("auto", "state", "circuit", "execution", "bloch"),
  size = c("double", "single"),
  width = NULL,
  height = NULL,
  units = c("mm", "in", "cm"),
  dpi = 450,
  theme = c("nature", "npj", "colorblind", "dark", "light", "mono"),
  step = NULL,
  top = NULL,
  qubit = 1L,
  main = NULL,
  subtitle = NULL,
  overwrite = FALSE
)

## S3 method for class 'qv_export'
print(x, ...)
```

Arguments

`x`	A circuit, statevector, simulation result, Bloch vector, or Bloch trajectory.
`file`	Destination ending in `.pdf`, `.svg`, `.png`, or `.tiff`.
`view`	Figure view. `"auto"` infers it from `x`.
`size`	Double-column (183 mm) or single-column (89 mm) width preset.
`width, height`	Optional custom dimensions interpreted in `units`.
`units`	Millimetres, inches, or centimetres.
`dpi`	Raster resolution for PNG and TIFF.
`theme`	Nature-style, npj-inspired, colorblind, dark, light, or monochrome preset.
`step`	Recorded step for an execution view.
`top`	Maximum basis states shown in state views.
`qubit`	One-based qubit index for a Bloch view.

main, subtitle	Optional figure title and subtitle.
overwrite	Replace an existing destination.
...	Additional print arguments reserved for future use.

Details

The journal presets follow Nature's public final-figure widths of 89 mm and 183 mm. PDF and SVG preserve vector artwork; PNG and TIFF default to 450 dpi.

Value

A qv_export describing the created artifact.

Examples

```
result <- quantum_circuit(1L) |>
  gate_h(1L) |>
  simulate_quantum(backend = "reference")
file <- tempfile(fileext = ".pdf")
save_quantum_plot(result, file, view = "state", size = "single")
invisible(unlink(file))
```

---

simulate_quantum *Simulate Quantum Circuits and Inspect State Data*

Description

Executes a validated circuit without constructing full-system gate matrices. Exact state probabilities and optional reproducible shot counts share one stable result schema. Before allocating the state, the function conservatively estimates memory for initialization, backend workspace, probability temporaries, shot sampling, and retained trajectory states. The default guard is exactly 2 GiB unless option qvivid.memory_limit_gib is set; a per-call memory_limit_gib value takes precedence.

Usage

```
simulate_quantum(
  circuit,
  shots = NULL,
  seed = NULL,
  initial_state = NULL,
  backend = c("auto", "native", "reference"),
  record = FALSE,
  memory_limit_gib = getOption("qvivid.memory_limit_gib", 2)
)

state_data(x, include_zero = TRUE, tolerance = 1e-14)
```

```
trajectory_data(result, include_zero = TRUE)

## S3 method for class 'qv_result'
print(x, ...)
```

Arguments

circuit	A qv_circuit.
shots	Optional positive number of terminal measurement samples.
seed	Optional non-negative integer seed.
initial_state	Optional normalized complex statevector.
backend	The automatic, native, or readable reference backend.
record	Retain state after every operation for animation.
memory_limit_gib	Maximum estimated peak memory, in GiB. Defaults to option qvivid.memory_limit_gib, or 2 GiB when the option is unset. The estimate includes the statevector, initialization and backend workspace, probability temporaries, shot sampling, and a recorded trajectory. Explicitly raise the value when sufficient memory is available, or use Inf to disable the guard.
x	A complex statevector or qv_result.
result	A qv_result created with record = TRUE.
include_zero	Include effectively zero-probability basis states.
tolerance	Threshold used to classify zero probabilities.
...	Additional arguments reserved for methods.

Value

simulate_quantum() returns a qv_result. The following named elements form the stable qvivid 0.1.x result schema:

circuit The input qv_circuit.

state The final complex statevector. Qubit 1 is the least-significant statevector bit; displayed basis strings put the highest-numbered qubit first.

probabilities Exact probabilities in the same order as state.

counts A data frame with character basis, integer count, and numeric probability columns. Basis strings put the highest-numbered classical bit first. With shots = NULL, this is a zero-row data frame with the same columns.

shots, seed The integer sampling inputs, or NULL.

backend The resolved backend name.

elapsed Elapsed simulation time in seconds.

trajectory NULL, or a list of frames containing integer step, character label, operation, and complex state fields.

schema_version The integer result-schema version.

state_data() returns stable columns index, basis, real, imaginary, magnitude, probability, and phase. The index is zero-based; phase is NA below tolerance. trajectory_data() appends integer step and character label columns to those state columns.

Examples

```
bell <- quantum_circuit(2) |>
  gate_h(1) |>
  gate_cx(1, 2) |>
  measure_all()

result <- simulate_quantum(
  bell,
  shots = 1000,
  seed = 42,
  backend = "reference",
  record = TRUE
)
state_data(result, include_zero = FALSE)
trajectory_data(result)
```

---

standard-gates *Add Quantum Gates to a Circuit*

Description

Adds standard or custom one- and two-qubit unitary operations. All functions accept the circuit first and therefore compose with the base R pipe.

Usage

```
gate_h(circuit, qubit)
gate_x(circuit, qubit)
gate_y(circuit, qubit)
gate_z(circuit, qubit)
gate_s(circuit, qubit)
gate_t(circuit, qubit)
gate_rx(circuit, qubit, theta)
gate_ry(circuit, qubit, theta)
gate_rz(circuit, qubit, theta)
gate_cx(circuit, control, target)
gate_cz(circuit, control, target)
gate_swap(circuit, qubit1, qubit2)
gate_unitary(circuit, matrix, qubits, label = "U")
```

Arguments

`circuit`	A `qv_circuit`.
`qubit`	A one-based qubit index.
`theta`	Rotation angle in radians.
`control`, `target`	Distinct one-based control and target indices.
`qubit1`, `qubit2`	Distinct one-based qubit indices.

matrix	A numeric or complex 2 by 2 or 4 by 4 unitary matrix.
qubits	One or two one-based qubit indices.
label	A short circuit-diagram label for a custom unitary.

Details

For a two-qubit custom gate on c(q1, q2), the supplied matrix uses local basis order |00⟩, |01⟩, |10⟩, |11⟩ with q1 as the first qubit.

Value

A modified qv_circuit.

Examples

```
bell <- quantum_circuit(2) |>
  gate_h(1) |>
  gate_cx(1, 2)
```

Index