

Package: qsplines (via r-universe)

August 28, 2024

Type Package

Title Quaternions Splines

Version 1.0.1

Description Provides routines to create some quaternions splines: Barry-Goldman algorithm, De Casteljaou algorithm, and Kochanek-Bartels algorithm. The implementations are based on the Python library 'splines'. Quaternions splines allow to construct spherical curves. References: Barry and Goldman <doi:10.1145/54852.378511>, Kochanek and Bartels <doi:10.1145/800031.808575>.

License GPL-3

URL <https://github.com/stla/qsplines>

BugReports <https://github.com/stla/qsplines/issues>

LinkingTo Rcpp, BH

Depends onion

Imports shiny, utils, Rcpp

Suggests rgl

Encoding UTF-8

RoxygenNote 7.2.3

Repository <https://stla.r-universe.dev>

RemoteUrl <https://github.com/stla/qsplines>

RemoteRef HEAD

RemoteSha 6e1b2f703c76d6383419a3695719e182155165a6

Contents

| | |
|------------------|---|
| BarryGoldman | 2 |
| DeCasteljau | 3 |
| interpolateTimes | 4 |
| KochanekBartels | 5 |
| quaternionFromTo | 7 |
| shinyKBS | 8 |

BarryGoldman

*Barry-Goldman quaternions spline***Description**

Constructs a spline of unit quaternions by the Barry-Goldman method.

Usage

```
BarryGoldman(keyRotors, keyTimes = NULL, n_intertimes, times)
```

Arguments

| | |
|--------------|---|
| keyRotors | a vector of unit quaternions (rotors) to be interpolated; it is automatically appended with the first one to have a closed spline |
| keyTimes | the times corresponding to the key rotors; must be an increasing vector of length <code>length(keyRotors)+1</code> ; if NULL, it is set to <code>c(1, 2, ..., length(keyRotors)+1)</code> |
| n_intertimes | a positive integer used to linearly interpolate the times given in keyTimes in order that there are <code>n_intertimes - 1</code> between two key times (so one gets the key times if <code>n_intertimes = 1</code>); if this argument is given, then it has precedence over times |
| times | the interpolating times, they must lie within the range of keyTimes; ignored if n_intertimes is given |

Value

A vector of unit quaternions with the same length as times.

Note

The function does not check whether the quaternions given in keyRotors are unit quaternions.

Examples

```
library(qsplines)
# Using a Barry-Goldman quaternions spline to construct
# a spherical curve interpolating some key points on
# the sphere of radius 5.

# helper function: spherical to Cartesian coordinates
sph2cart <- function(rho, theta, phi){
  return(c(
    rho * cos(theta) * sin(phi),
    rho * sin(theta) * sin(phi),
    rho * cos(phi)
  ))
}
```

```

# construction of the key points on the sphere
keyPoints <- matrix(nrow = 0L, ncol = 3L)
theta_ <- seq(0, 2*pi, length.out = 9L)[-1L]
phi <- 1
for(theta in theta_){
  keyPoints <- rbind(keyPoints, sph2cart(5, theta, phi))
  phi = pi - phi
}
n_keyPoints <- nrow(keyPoints)

# construction of the key rotors; the first key rotor is the
# identity quaternion and rotor i sends the first key point
# to the key point i
keyRotors <- quaternion(length.out = n_keyPoints)
rotor <- keyRotors[1L] <- H1
for(i in seq_len(n_keyPoints - 1L)){
  keyRotors[i+1L] <- rotor <-
    quaternionFromTo(
      keyPoints[i, ]/5, keyPoints[i+1L, ]/5
    ) * rotor
}

# Barry-Goldman quaternions spline
rotors <- BarryGoldman(keyRotors, n_intertimes = 10L)

# construction of the interpolating points on the sphere
points <- matrix(nrow = 0L, ncol = 3L)
keyPoint1 <- rbind(keyPoints[1L, ])
for(i in seq_along(rotors)){
  points <- rbind(points, rotate(keyPoint1, rotors[i]))
}

# visualize the result with the 'rgl' package
library(rgl)
spheres3d(0, 0, 0, radius = 5, color = "lightgreen")
spheres3d(points, radius = 0.2, color = "midnightblue")
spheres3d(keyPoints, radius = 0.25, color = "red")

```

DeCasteljau

Spline using the De Casteljau algorithm

Description

Constructs a quaternions spline using the De Casteljau algorithm.

Usage

```

DeCasteljau(
  segments,

```

```

    keyTimes = NULL,
    n_intertimes,
    times,
    constantSpeed = FALSE
)

```

Arguments

| | |
|---------------|---|
| segments | a list of vectors of unit quaternions; each segment must contain at least two quaternions |
| keyTimes | the times corresponding to the segment boundaries, an increasing vector of length $\text{length}(\text{segments})+1$; if NULL, it is set to 1, 2, ..., $\text{length}(\text{segments})+1$ |
| n_intertimes | a positive integer used to linearly interpolate the times given in keyTimes in order that there are $n_intertimes - 1$ between two key times (so one gets the key times if $n_intertimes = 1$); this parameter must be given if $\text{constantSpeed}=\text{TRUE}$ and if it is given when $\text{constantSpeed}=\text{FALSE}$, then it has precedence over times |
| times | the interpolating times, they must lie within the range of keyTimes; ignored if $\text{constantSpeed}=\text{TRUE}$ or if $n_intertimes$ is given |
| constantSpeed | Boolean, whether to re-parameterize the spline to have constant speed; in this case, "times" is ignored and a function is returned, with an attribute "times", the vector of new times corresponding to the key rotors |

Value

A vector of quaternions whose length is the number of interpolating times.

Note

This algorithm is rather for internal purpose. It serves for example as a base for the [Konachek-Bartels](#) algorithm.

| | |
|------------------|--------------------------------------|
| interpolateTimes | <i>Interpolate a vector of times</i> |
|------------------|--------------------------------------|

Description

Linearly interpolate an increasing vector of times. This is useful to deal with the quaternions splines.

Usage

```
interpolateTimes(times, n, last = TRUE)
```

Arguments

| | |
|-------|--|
| times | increasing vector of times |
| n | integer, controls the number of interpolations: there will be n-1 time values between two consecutive original times |
| last | Boolean, whether to include or exclude the last element |

Value

A vector, a refinement of the times vector.

Examples

```
library(qsplines)
interpolateTimes(1:4, n = 3)
interpolateTimes(c(1, 2, 4), n = 3)
```

KochanekBartels

Kochanek-Bartels quaternions spline

Description

Constructs a quaternions spline by the Kochanek-Bartels algorithm.

Usage

```
KochanekBartels(
  keyRotors,
  keyTimes = NULL,
  tcb = c(0, 0, 0),
  times,
  n_intertimes,
  endcondition = "natural",
  constantSpeed = FALSE
)
```

Arguments

| | |
|-----------|---|
| keyRotors | a vector of unit quaternions (rotors) to be interpolated |
| keyTimes | the times corresponding to the key rotors; must be an increasing vector of the same length a keyRotors if endcondition = "natural" or of length one more than number of key rotors if endcondition = "closed" |
| tcb | a vector of three numbers respectively corresponding to tension, continuity and bias |
| times | the times of interpolation; each time must lie within the range of the key times; this parameter can be missing if keyTimes is NULL and n_intertimes is not missing, and it is ignored if constantSpeed=TRUE |

`n_intertimes` if given, this argument has precedence over `times`; `keyTimes` can be `NULL` and `times` is constructed by linearly interpolating the key times such that there are `n_intertimes - 1` between two key times (so the times are the key times if `n_intertimes = 1`)

`endcondition` start/end conditions, can be "closed" or "natural"

`constantSpeed` Boolean, whether to re-parameterize the spline to have constant speed; in this case, "times" is ignored and you must set the interpolating times with the help of `n_intertimes`

Value

A vector of quaternions having the same length as the `times` vector.

Examples

```
library(qsplines)
# Using a Kochanek-Bartels quaternions spline to construct
# a spherical curve interpolating some key points on the
# sphere of radius 5

# helper function: spherical to Cartesian coordinates
sph2cart <- function(rho, theta, phi){
  return(c(
    rho * cos(theta) * sin(phi),
    rho * sin(theta) * sin(phi),
    rho * cos(phi)
  ))
}

# construction of the key points on the sphere
keyPoints <- matrix(nrow = 0L, ncol = 3L)
theta_ <- seq(0, 2*pi, length.out = 9L)[-1L]
phi <- 1.3
for(theta in theta_){
  keyPoints <- rbind(keyPoints, sph2cart(5, theta, phi))
  phi = pi - phi
}
n_keyPoints <- nrow(keyPoints)

# construction of the key rotors; the first key rotor
# is the identity quaternion and rotor i sends the
# first key point to the i-th key point
keyRotors <- quaternion(length.out = n_keyPoints)
rotor <- keyRotors[1L] <- H1
for(i in seq_len(n_keyPoints - 1L)){
  keyRotors[i+1L] <- rotor <-
    quaternionFromTo(
      keyPoints[i, ]/5, keyPoints[i+1L, ]/5
    ) * rotor
}
```

```

# Kochanek-Bartels quaternions spline
rotors <- KochanekBartels(
  keyRotors, n_intertimes = 25L,
  endcondition = "closed", tcb = c(-1, 5, 0)
)

# construction of the interpolating points on the sphere
points <- matrix(nrow = 0L, ncol = 3L)
keyPoint1 <- rbind(keyPoints[1L, ])
for(i in seq_along(rotors)){
  points <- rbind(points, rotate(keyPoint1, rotors[i]))
}

# visualize the result with the 'rgl' package
library(rgl)
spheres3d(0, 0, 0, radius = 5, color = "lightgreen")
spheres3d(points, radius = 0.2, color = "midnightblue")
spheres3d(keyPoints, radius = 0.25, color = "red")

```

quaternionFromTo *Quaternion between two vectors*

Description

Get a unit quaternion whose corresponding rotation sends u to v ; the vectors u and v must be normalized.

Usage

```
quaternionFromTo(u, v)
```

Arguments

u, v two unit 3D vectors

Value

A unit quaternion whose corresponding rotation transforms u to v .

Examples

```

library(qsplines)
u <- c(1, 1, 1) / sqrt(3)
v <- c(1, 0, 0)
q <- quaternionFromTo(u, v)
rotate(rbind(u), q) # this should be v

```

`shinyKBS`*Shiny demonstration of Kochanek-Bartels spline*

Description

Run a Shiny app which demonstrates the Kochanek-Bartels spline.

Usage

```
shinyKBS()
```

Value

No value returned.

Index

BarryGoldman, [2](#)

DeCasteljau, [3](#)

interpolateTimes, [4](#)

KochanekBartels, [5](#)

Konachek-Bartels, [4](#)

quaternionFromTo, [7](#)

shinyKBS, [8](#)