Interface Shiny pour l'analyse et la visualisation de séries temporelles de croissances de plantes

Mathis CORDIER

2019/2020

Sommaire

Objectifs du projet

L'objectif est de permettre à un utilisateur de traiter ses **séquences d'images** afin d'**extraire les informations** importantes sur la **croissance des plantes**.

On crée alors une interface mettant à disposition :

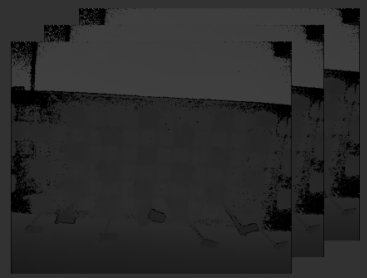
- **Pipelines** pour l'analyse de séquences d'images
- **Visualisations** pour caractérisation de croissance
- Possibilité de récupération de données traitées

Données

Nous disposons de séquences d'images RGBD :

Red - **G**reen - **B**lue - **D**epth

Données :



Construction de l'interface

Le travail effectué lors du projet se résume chronologiquement par les étapes suivantes :

- Création de l'**architecture** de l'interface
- Recherche de moyens d'**extraction** de l'information
- Création des fonctions de **visualisation**
- **Développement** de l'interface
- **Intégration** des fonctions de visualisation à l'interface
- **Hébergement** de l'interface

Moyens d'extraire l'information

Comment caractériser la croissance des plantes ?

- **RGB** :
  - Évolution de la couleur moyenne de la canopée :
    i. Calcul du masque de segmentation binaire
    ii. Application du masque à l'image
    iii. Calcul de la couleur moyenne
  - Évolution de la surface de la canopée vue de dessus :
    i. Calcul du masque de segmentation binaire
    ii. Calcul de la surface allumée du masque
  - Comparaison de la surface entre 2 instants :
    i. Calcul du masque de segmentation binaire
    ii. Différence entre les masques

Moyens d'extraire l'information

Comment caractériser la croissance des plantes ?

- **Depth** :
  - Évolution de la distance moyenne à la caméra :
    i. Calcul de la moyenne des niveaux de gris
  - Analyse fréquentielle de la série temporelle :
    - Calcul de la fréquence fondamentale du signal :

$$c_u(n) = \frac{2}{T}\sqrt{\left(\sum_{k=nT}^{(n+1)T-1} u(k,n)\cos(\frac{2k\pi}{T})\right)^2 + \left(\sum_{k=nT}^{(n+1)T-1} u(k,n)\sin(\frac{2k\pi}{T})\right)^2}$$

    - Calcul de l'énergie du signal :

$$E_u(n) = \sum_{k=nT}^{(n+1)T-1} u(k,n)^2$$

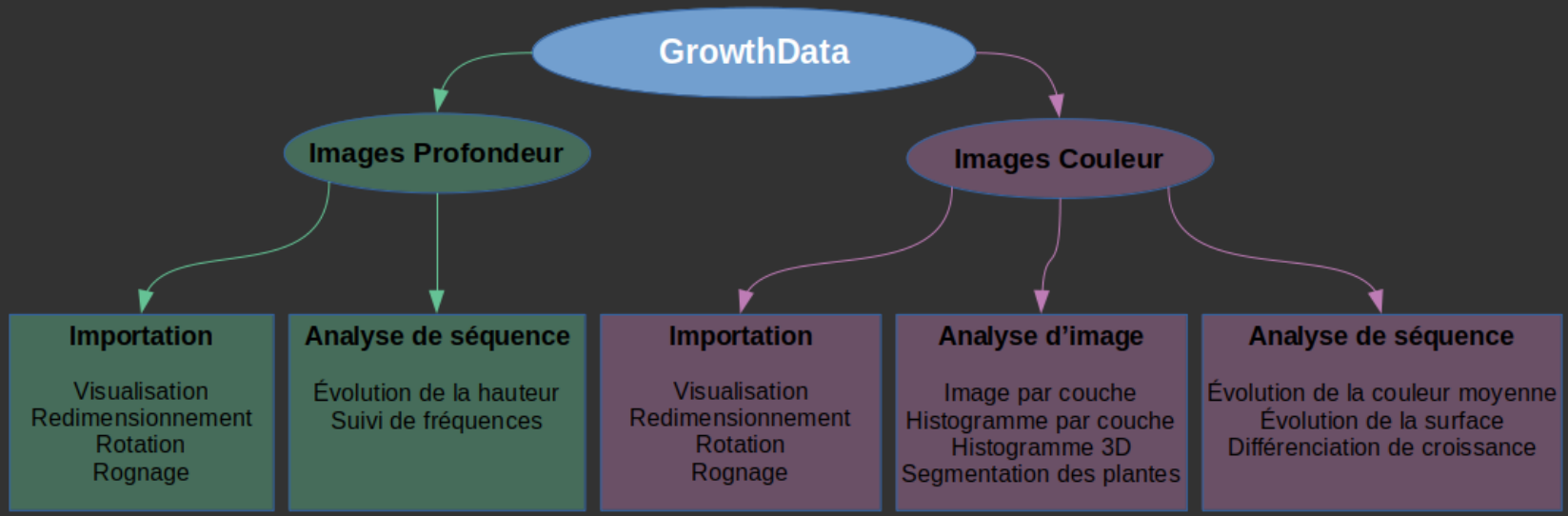
    - Calcul du taux de distorsion harmonique :

$$HDR_u(n) = 100 \times \sqrt{\frac{E(n) - \frac{1}{2}c(n)^2}{\frac{1}{2}c(n)^2}}$$

    - Calcul de la pente moyenne :

Par régression linéaire sur $u(\cdot, n)$

Schéma de l'interface



Code de l'application - Packages

Packages utilisés :

```r
# SHINY
library(shiny)
library(shinydashboard)
library(shinyWidgets)
library(shinycssloaders)

# IMAGES
library(ijtiff)
library(raster)
library(imager)

# VISUALISATION
library(ggplot2)
library(hrbrthemes)
library(plotly)
library(scales)

# DATA
library(data.table)
```

Code de l'application - Préambule

Par défaut, *Shiny* limite la taille des fichiers importés à 5 Mo. L'importation de séquences d'images nécessite une limite plus élevée, ce qui donne lieu à la commande suivante :

```
options(shiny.maxRequestSize=2000*1024^2) # Import file max size : 2GB
```

Code de l'application - Préambule

Supposons que l'on importe une séquence de 9 images couleur (RGB) au format Full HD (1920x1080) codées sur 8 bits (1 octet), nommée *sequence*.
La séquence sera donc vue comme un *array* à 4 dimensions.

Sa dimension :

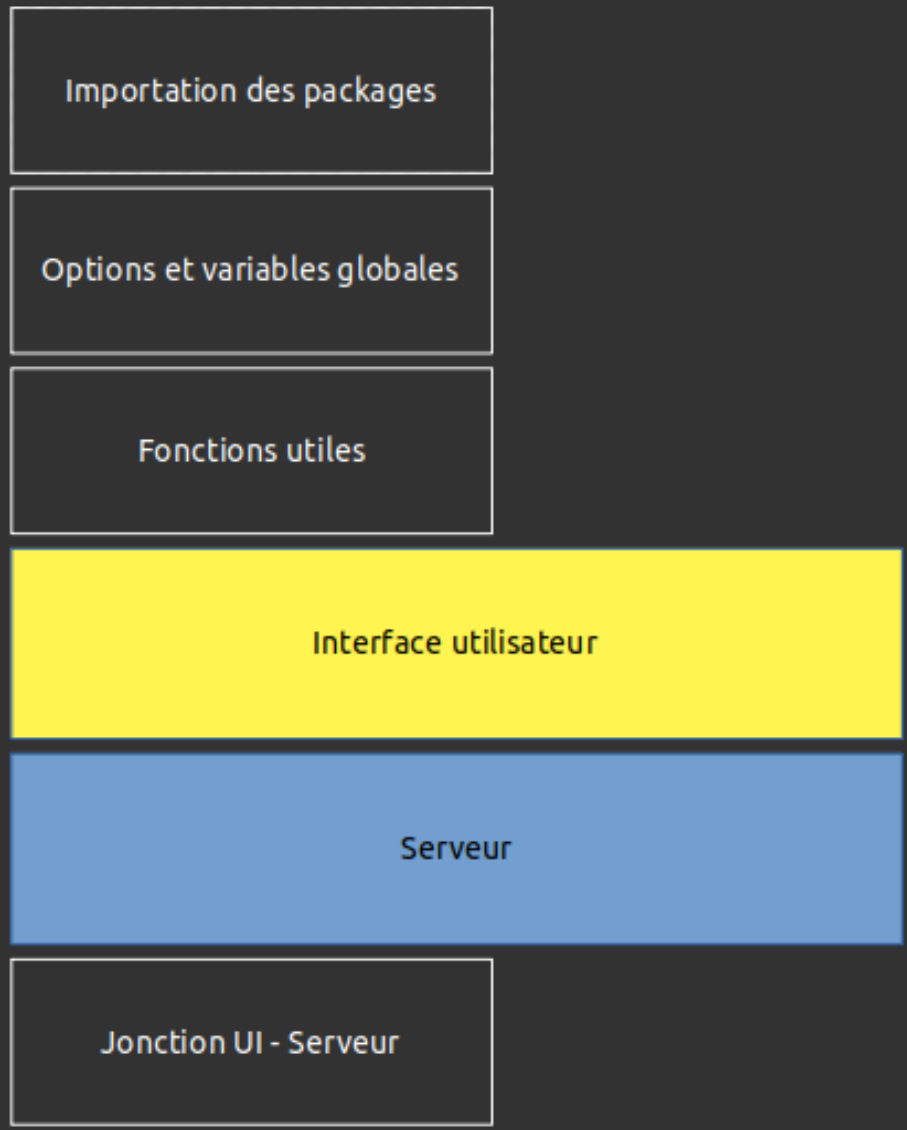
```
> dim(sequence)
[1] 1080 1920    3    9
```

Son poids :

```
> format(object.size(sequence)/8,units="MB",standard="SI")
[1] "56 MB"
```

Code de l'application - Interface Shiny

Structure du code :



Code de l'application - Interface Shiny

Interface utilisateur	Serveur	Jonction

`ui <- dashboardPage(`
`  skin = "blue",`
`  title = "GrowthData",`
`  dashboardHeader(...),`
`  dashboardSidebar(`
`  sidebarMenu(`
`  id = "id_menu",`
`  menuItem("A"),`
`  menuItem("B"),`
`  menuItem("C"),`
`  ...`
`  )`
`  ),`
`  dashboardBody(`
`  tabItems(`
`  tabItem(tabName="A",`
`  ...),`
`  tabItem(tabName="B",`
`  ...),`
`  tabItem(tabName="C",`
`  ...),`
`  ...`
`  )`
`  )`
`)`
	`server = function(input, output, session) {`
	`  input$...`
	`  output$...`
	`  ...`
	`}`
		`shinyApp(ui,server)`

Code de l'application - Interface Shiny

Interface utilisateur	Serveur
numericInput sliderInput sliderTextInput pickerInput fileInput	input$id_input
actionButton	observeEvents(input$id_button, ...)
uiOutput htmlOutput	output$id_ui = renderUI(...)
plotOutput	output$id_plot = renderPlot(...)
plotlyOutput	output$id_plotly = renderPlotly(...)
downloadButton	output$id_button = downloadHandler(...)

On peut imbriquer plusieurs *uiOutput* afin de maîtriser le placement des différentes frames ou réagir à un évènement.

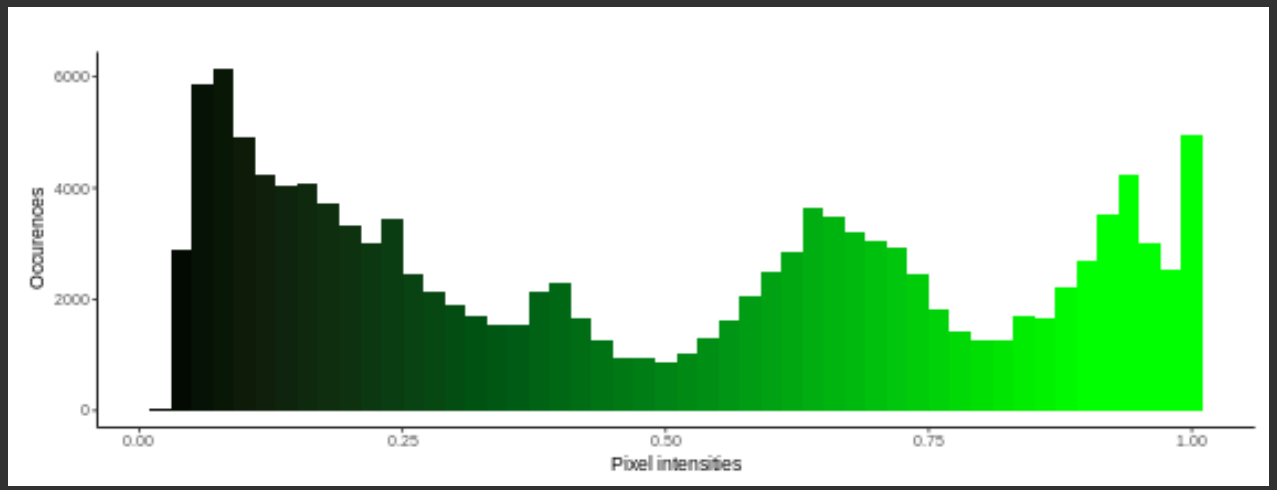
Code de l'application - Quelques fonctions de visualisation

Histogramme d'une couche couleur :

```
decompimgNG = function(mat,col_couche){
  df = data.frame(X=as.numeric(mat))
  colnames(df) = "X"
  p = ggplot(df,aes(x=X,fill=..x..)) + geom_histogram(bins=50) + theme_classic() +
    scale_fill_gradient(low="black",high=col_couche) +
    theme(legend.position="none") + labs(x="Pixel intensities", y="Occurences")
  return(p)
}
```

Code de l'application - Quelques fonctions de visualisation

Histogramme d'une couche couleur :



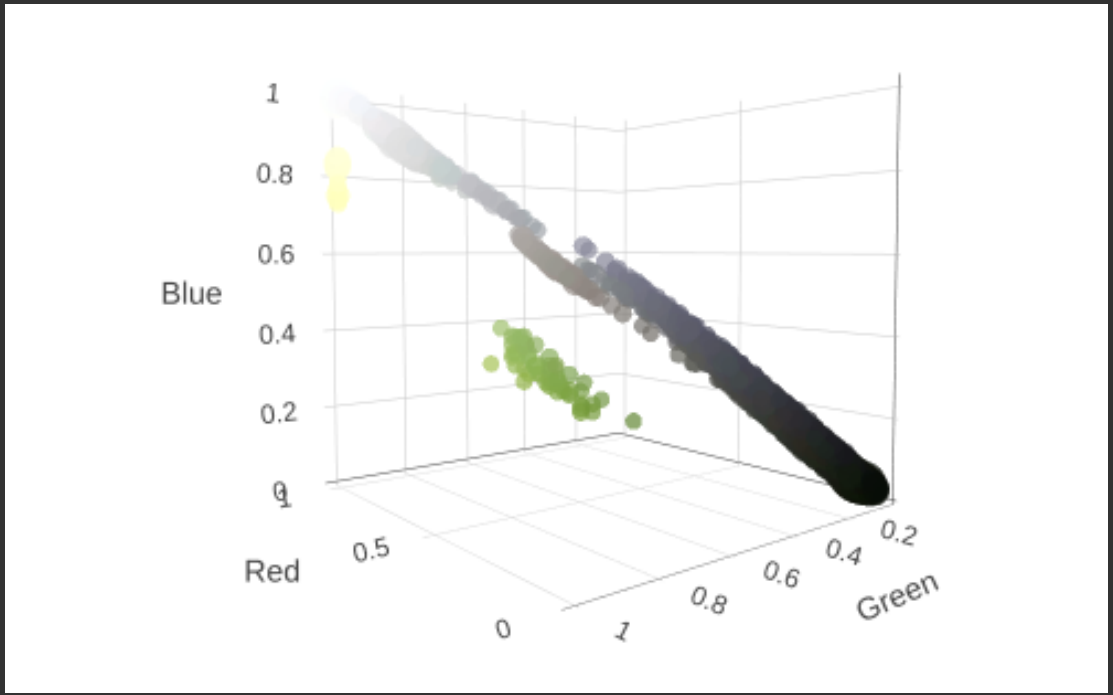
Code de l'application - Quelques fonctions de visualisation

Histogramme 3D de décomposition d'image :

```
decompimgRGB = function(im,threshold){
  df = as.data.frame(cbind(as.numeric(im[,,1]),as.numeric(im[,,2]),as.numeric(im[,,3])))
  colnames(df) = c("R","G","B")
  df = setDT(df)[, .N, by = c("R","G","B")]
  df = df[df$N>threshold,]
  col = apply(df,1,RVB)
  df = as.data.frame(cbind(df,col))
  p = plot_ly(df, x=~R, y=~G, z=~B,
          text=~paste("Color : ",col,"<br> R:",R,",G:",G,",B:",B,"<br> Number of occurences : ",N),
          hoverinfo = 'text',
          marker = list(size = ~2*log(N+1),
                        color = ~col,
                        line = list(width = 0))) %>%
      add_markers() %>%
      layout(title="3D Histogram",
           scene = list(xaxis = list(title = 'Red'),
                        yaxis = list(title = 'Green'),
                        zaxis = list(title = 'Blue'))
    )
  return(p)
}
```

Code de l'application - Quelques fonctions de visualisation

Histogramme 3D de décomposition d'image :



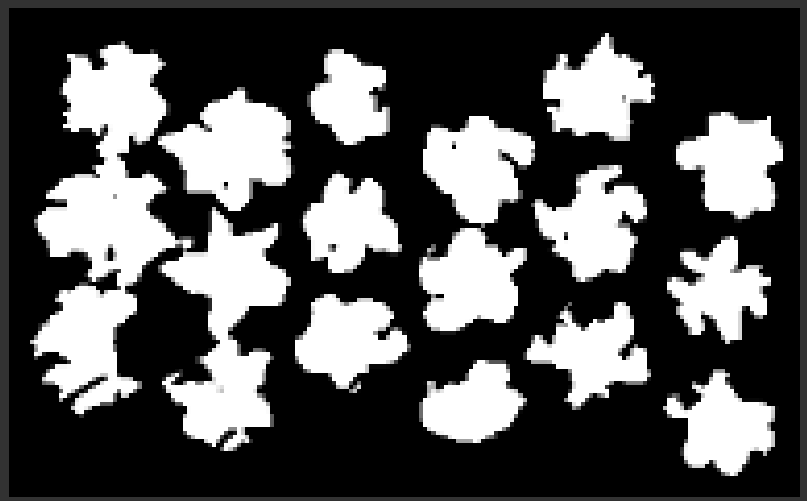
Code de l'application - Quelques fonctions de visualisation

Segmentation des plantes :

```
segmentation = function(im,bin=TRUE){
  img = 2*(im-0.4)
  im_2grb = 2*img[,,2]-img[,,1]-img[,,3]
  im_gr = img[,,2]-img[,,1]
  im_2grb[im_2grb<0] = 0
  im_2grb[im_2grb>1] = 1
  im_gr[im_gr<0] = 0
  im_gr[im_gr>1] = 1
  seuil_2grb = 0.6
  seuil_gr_min = 0.04
  seuil_gr_max = 0.4
  cond = im_2grb>seuil_2grb & im_gr>seuil_gr_min & im_gr<seuil_gr_max
  if (bin) {return(cond)}
  else {
    for (i in 1:3){
      im[,,i][not(cond)]=0
    }
    return(im)
  }
}
```

Code de l'application - Quelques fonctions de visualisation

Segmentation des plantes :



Démonstration de l'application

Au choix :

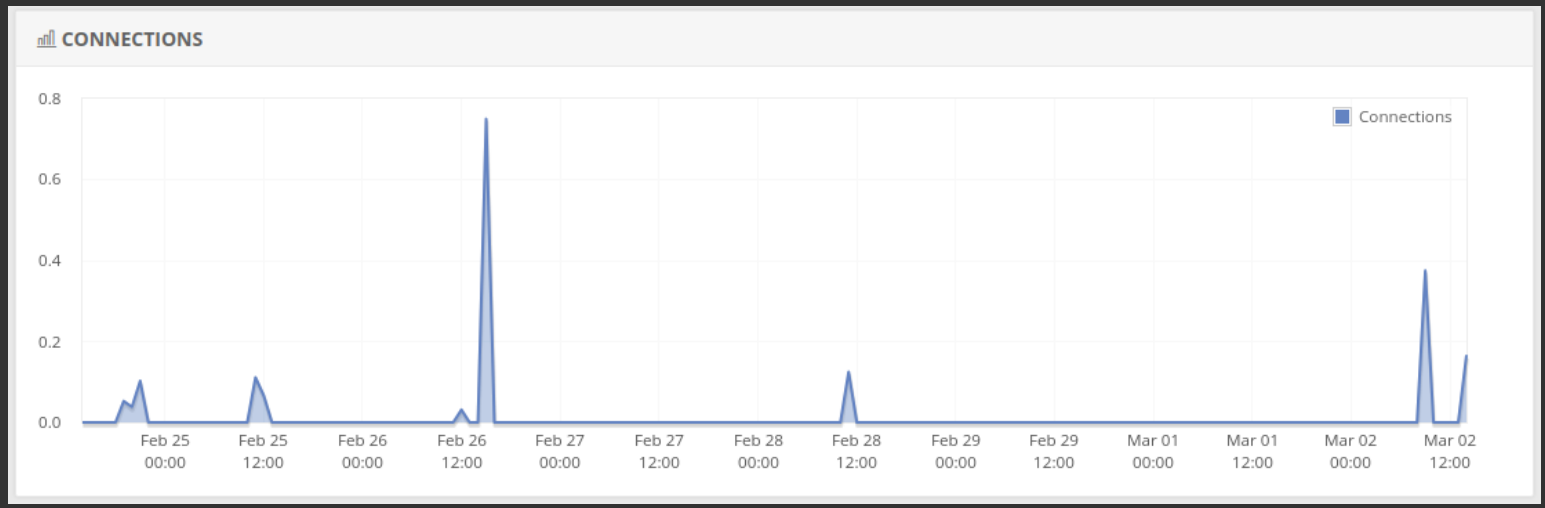
- En local :
  - Plus rapide
  - Utilise à 100% les capacités de la machine
  - Nécessite d'avoir l'application sur disque dur (légère), mais surtout R et les packages requis (lourd)
  - Accès au code
- Sur `https://sithamfr.shinyapps.io/GrowthData/` :
  - Plus lent
  - Utilise principalement les capacités du serveur
  - Ne nécessite aucun fichier sur le disque dur, ni R
  - Code reste privé

Statistiques d'utilisation

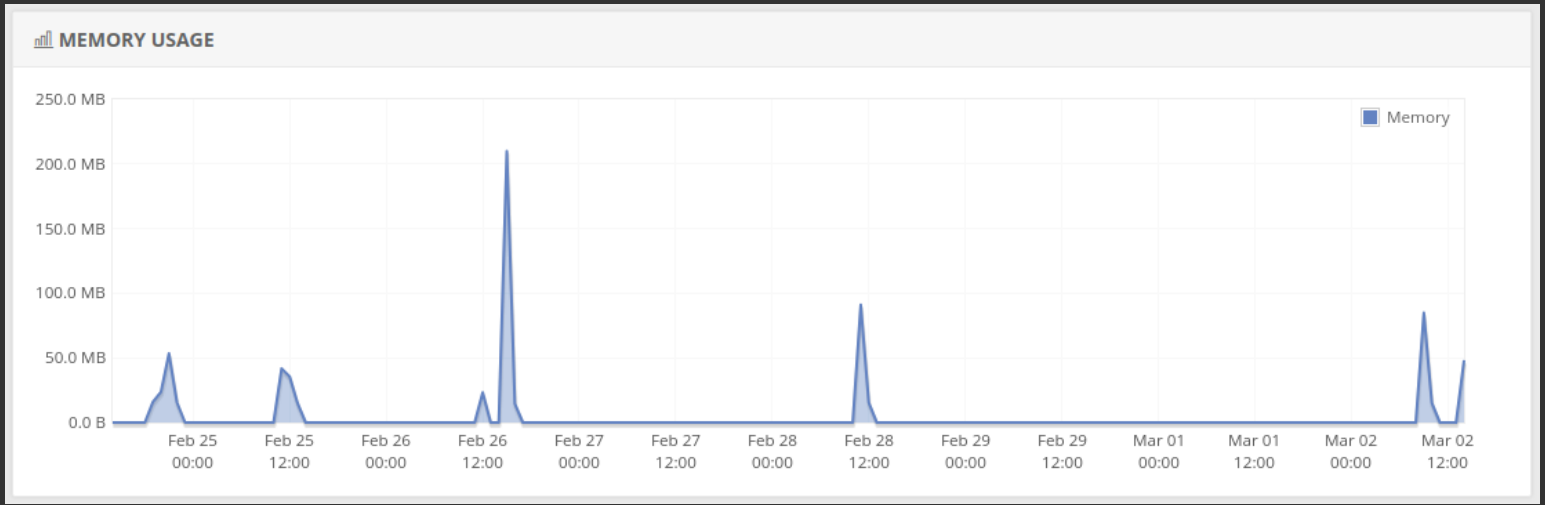
Shinyapps.io permet au créateur de l'application d'avoir des retours sur son utilisation :

- Temps de connexion par tranche horaire
- Mémoire utilisée par tranche horaire
- Temps d'utilisation du processeur par tranche horaire
- Taille des transferts machine/serveur par tranche horaire
- Fichier Log des exécutions de l'application

Statistiques d'utilisation - Temps de connexion

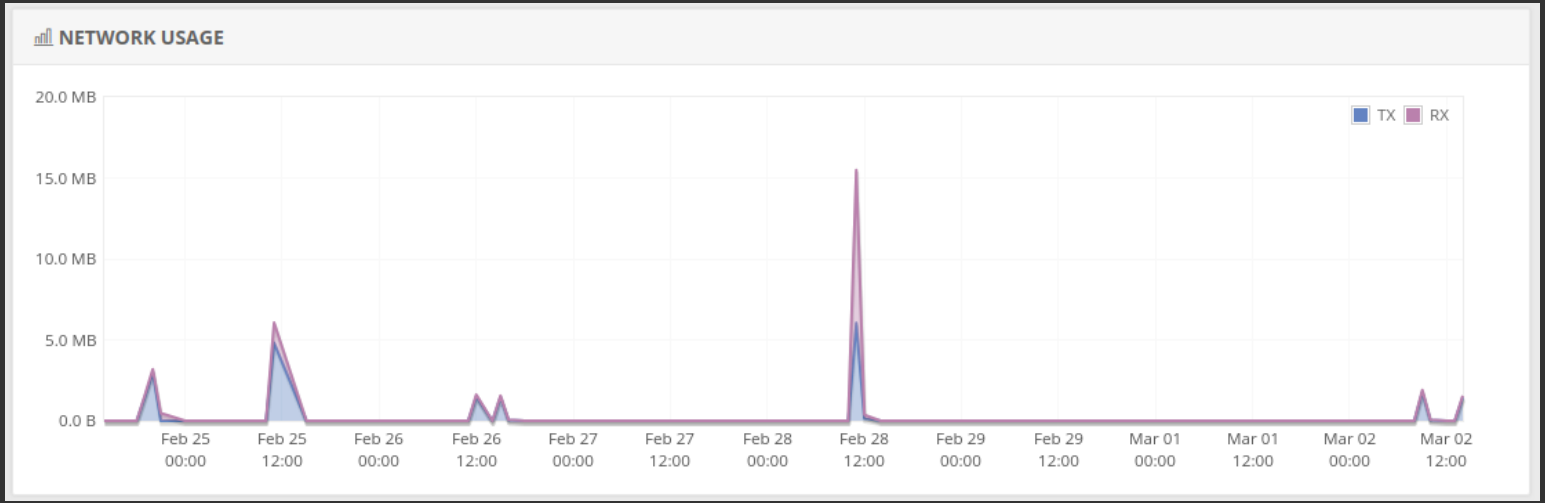


Statistiques d'utilisation - Mémoire



Statistiques d'utilisation - Temps d'utilisation du processeur

Statistiques d'utilisation - Taille des transferts



Statistiques d'utilisation - Fichier Log

```
2020-03-02T14:03:46.361208+00:00 shinyapps[1708563]:
2020-03-02T14:03:46.362327+00:00 shinyapps[1708563]:
2020-03-02T14:03:46.362573+00:00 shinyapps[1708563]:
2020-03-02T14:03:46.362572+00:00 shinyapps[1708563]: The following object is masked from ‘package:graphics’:
2020-03-02T14:03:46.361535+00:00 shinyapps[1708563]: The following object is masked from ‘package:ggplot2’:
2020-03-02T14:03:46.373029+00:00 shinyapps[1708563]:
2020-03-02T14:03:46.361536+00:00 shinyapps[1708563]:     last_plot
2020-03-02T14:03:46.373031+00:00 shinyapps[1708563]:
2020-03-02T14:03:46.361535+00:00 shinyapps[1708563]:
2020-03-02T14:03:46.373031+00:00 shinyapps[1708563]: Attachement du package : ‘data.table’
2020-03-02T14:03:46.361790+00:00 shinyapps[1708563]: The following object is masked from ‘package:imager’:
2020-03-02T14:03:46.361791+00:00 shinyapps[1708563]:
2020-03-02T14:03:46.361791+00:00 shinyapps[1708563]:     highlight
2020-03-02T14:03:46.361792+00:00 shinyapps[1708563]:
2020-03-02T14:03:46.362058+00:00 shinyapps[1708563]: The following object is masked from ‘package:raster’:
2020-03-02T14:03:46.362059+00:00 shinyapps[1708563]:
2020-03-02T14:03:46.362060+00:00 shinyapps[1708563]:     select
2020-03-02T14:03:46.362060+00:00 shinyapps[1708563]:
2020-03-02T14:03:46.362325+00:00 shinyapps[1708563]: The following object is masked from ‘package:stats’:
2020-03-02T14:03:46.362326+00:00 shinyapps[1708563]:
2020-03-02T14:03:46.362326+00:00 shinyapps[1708563]:     filter
2020-03-02T14:03:46.373306+00:00 shinyapps[1708563]:
2020-03-02T14:03:46.361536+00:00 shinyapps[1708563]:
2020-03-02T14:03:46.373306+00:00 shinyapps[1708563]: The following object is masked from ‘package:raster’:
2020-03-02T14:03:46.373307+00:00 shinyapps[1708563]:
2020-03-02T14:03:46.373307+00:00 shinyapps[1708563]:     shift
2020-03-02T14:03:46.412617+00:00 shinyapps[1708563]:
2020-03-02T14:03:46.362573+00:00 shinyapps[1708563]:     layout
2020-03-02T14:03:46.412618+00:00 shinyapps[1708563]: Listening on http://127.0.0.1:36429
2020-03-02T14:03:46.362573+00:00 shinyapps[1708563]:
```

On peut voir ici les éventuelles erreurs auxquelles ce sont confrontés les utilisateurs, voire une erreur *out of memory*.