

R Google Earth Engine - Vol. 1



André Luiz Lima Costa

Licença: CC BY 4.0

RGEe foi criado por:

C Aybar, Q Wu, L Bautista, R Yali and A Barja (2020) rgee: An R package for interacting with Google Earth Engine Journal of Open
Source Software URL <https://github.com/r-spatial/rgee/>

Índice

Licença: CC BY 4.0.....	1
INSTALANDO.....	3
O FEIJÃO COM ARROZ.....	4
DADOS DE OUTRAS FONTES.....	9
UMA NOTA SOBRE COMO O GOOGLE EARTH ENGINE FUNCIONA.....	14
ee\$image.....	21
Construtores do ee\$image.....	22
Visualização de ee\$image.....	27
Informações e metadata do ee\$image.....	31
Operações aritméticas com ee\$image.....	34
Operações relacionais, boolianas e condicionais com ee#Image.....	36
Transformações espectrais com ee\$image.....	38
Gradiente de ee\$image.....	41
Convolução e detecção de bordas para ee\$image.....	43
ee\$imageCollection.....	50
Construtores de ImageCollection.....	50
Filtrando ImageCollection.....	52
Informações e metadata de ImageCollection.....	52
Visualizações com ImageCollection.....	53
Técnicas avançadas de Visualização de ImageCollection.....	55
ee\$Geometry.....	58
Planar x Geodésico.....	60
Informações e metadados de ee\$Geometry.....	61
Operações com ee\$Geometry.....	62
ee\$Feature.....	65
ee\$FeatureCollection.....	66
Visualização de ee\$FeatureCollection.....	68
Informações e Metadados de ee\$FeatureCollection.....	72
Filtrando ee\$FeatureCollection.....	74
Mapeando um ee\$FeatureCollection.....	74

VISITE:

amazeone.com.br

INSTALANDO

RGEE é um elo entre R e o Google Earth Engine que de forma bem simples e eficiente possibilita o acesso às principais capacidades desta ferramenta de dados geoespaciais.

Vamos aqui falar sobre as funções básicas como um ponto de partida. Primeiro temos que instalar os pacotes `sf` e o `mapview` (caso ainda não esteja presente no seu sistema). Da mesma forma instale o pacote `remotes`.

```
install.packages('sf')
install.packages('mapview')
install.packages('remotes')
```

Agora vamos instalar o `rgee`, lembrando que você precisará de ter uma conta ativa do Google Earth Engine no processo de instalação.

```
library(remotes)
install_github("r-spatial/rgee")
```

Agora é executar o comando seguinte somente uma vez para finalizar a instalação.

```
library(rgee)
ee_install()
```

E execute o comando de inicialização. Aqui você será direcionado a sua conta do Earth Engine para entrar no `rgee`.

```
ee_Initialize()
```

Pronto!!!

O FEIJÃO COM ARROZ

Vamos aqui ver como executar os processos mais básicos usando o rgee e imagem Sentinel2. Primeiro iniciamos o rgee com:

```
library(rgee)
ee_Initialize()
-- rgee 0.6.1 ----- earthengine-api 0.1.225 --
email: not_defined
Initializing Google Earth Engine: DONE!
Earth Engine user: users/xxxxxxxx
```

Agora vamos carregar uma imagem Sentinel2 e criar o primeiro mapa com um composição de cor verdadeira com as bandas 4, 3 e 2 nos canais RGB.

Carregando a imagem definida pela coordenada do ponto (longitude, latitude) e na data escolhida por início e fim.

```
#-----
#-- Selecionando a Imagem
#-----
coleção<-ee$ImageCollection('COPERNICUS/S2_SR')
ponto <- ee$Geometry$Point(-44.366,-18.145)
início <- ee>Date("2019-07-11")
fim <- ee>Date("2019-07-20")
filtro<-coleção$filterBounds(ponto)$filterDate(início,fim)
img <- filtro$first()
```

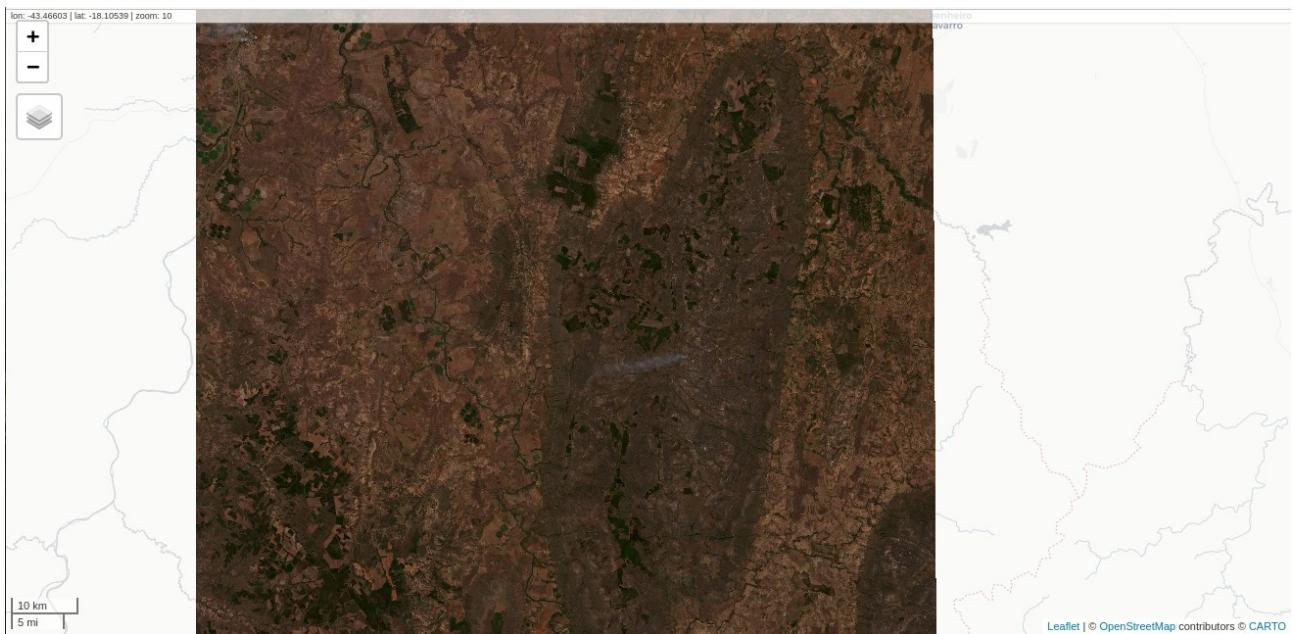
Criando o parâmetro de visualização escolhendo as bandas, o valor mínimo que será associado a 0 e o valor máximo que será associado a 1 e a correção gama para cada uma das três bandas.

```
#-----
# Combinação RGB Visível
#-----
vPar <- list(bands = c("B4", "B3", "B2"),min = 100,max = 8000,
              gamma = c(1.9,1.7,1.7))
```

E finalmente definindo a coordenada do ponto central do mapa e o nível de zoom do mapa a ser plotado.

```
Map$setCenter(-44.366,-17.69, zoom = 10)
Map$addLayer(img, vPar, "TCI")
```

O resultado será:

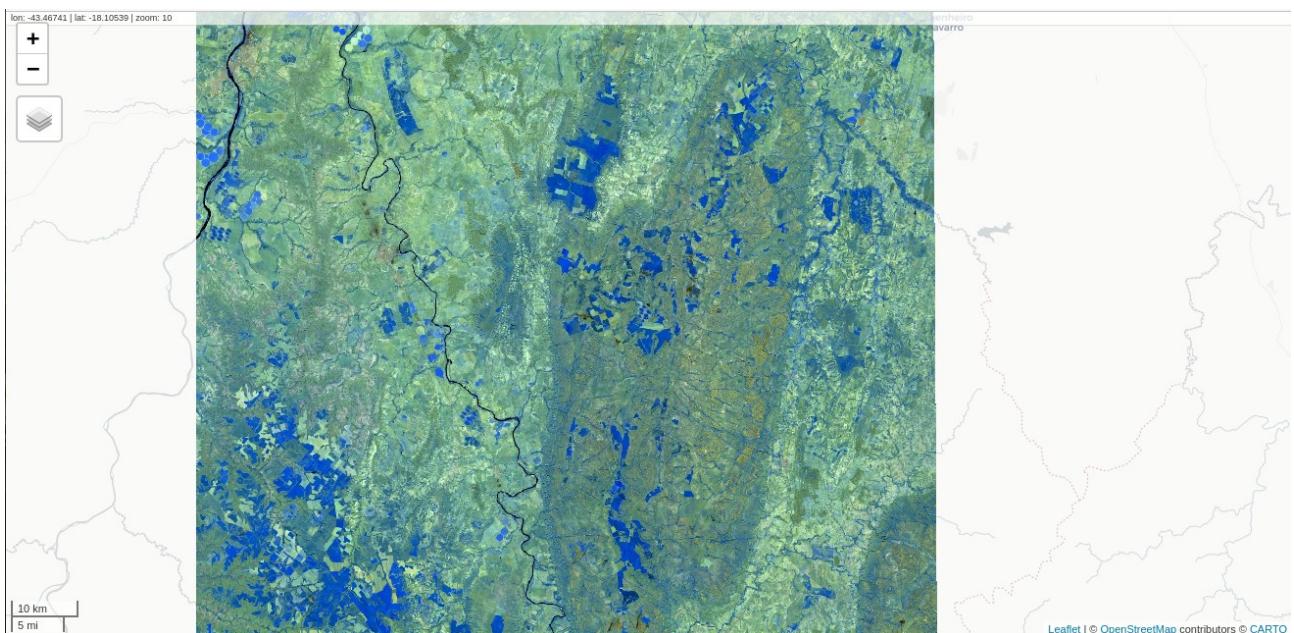


Vamos agora criar uma composição de cor falsa usando as bandas SWIR 12 e 11 a a banda NIR 8 nos canais RGB respectivamente. Esta composição ressalta rochas e solo.

```
#-----
# Combinação RGB Geologia e solo
#-----
vPar2 <- list(bands = c("B12", "B11", "B8"), min = 500, max = 5000,
               gamma = 1.6)
```

e plotamos com:

```
Map$addLayer(img, vPar2, "Geologia/Solo")
```



Agora vamos criar três índices normalizados (NDVI, NDWI e NDMI) que são índices de vegetação, de água e de umidade.

```
#-----
# Criando imagem NDVI
#-----
getNDVI <- function(image) {
  return(image$normalizedDifference(c("B8", "B4")))
}
ndvi1 <- getNDVI(img)
ndviPar <- list(palette = c(
  "#cccccc", "#f46d43", "#fdad61", "#fee08b",
  "#d9ef8b", "#a6d96a", "#66bd63", "#1a9850"
),min=0,max=1)

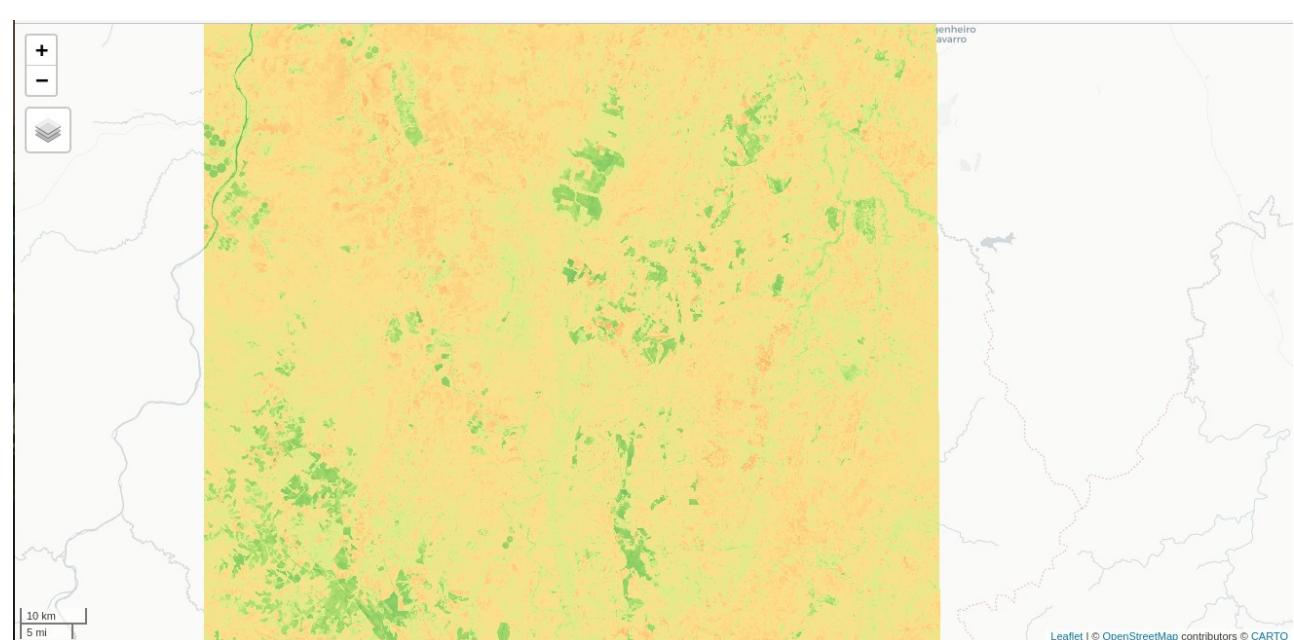
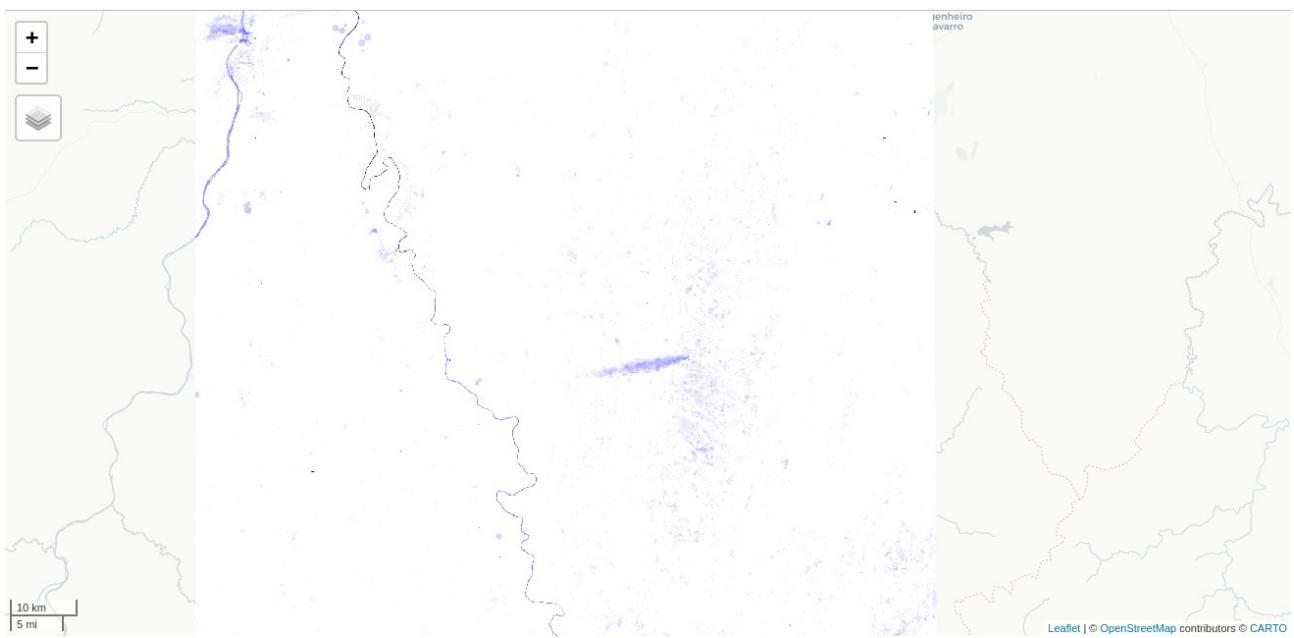
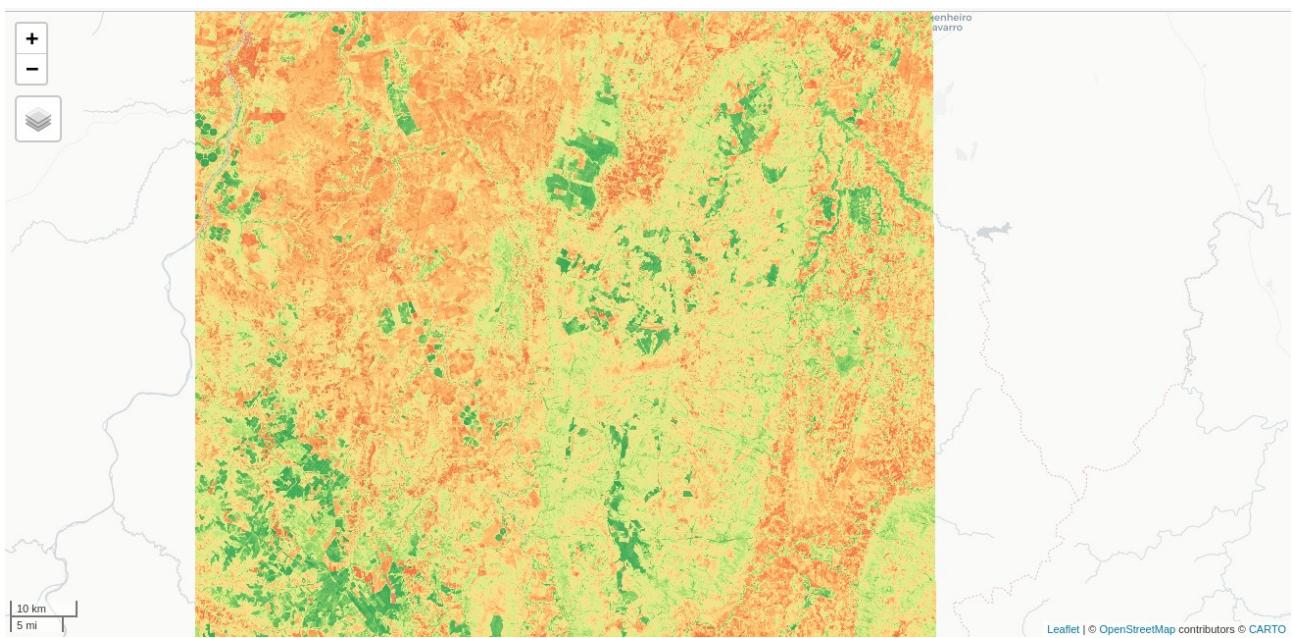
#-----
# Criando imagem NDWI
#-----
getNDWI <- function(image) {
  return(image$normalizedDifference(c("B3", "B5")))
}
ndwi <- getNDWI(img)
ndwiPar <- list(palette = c("#ffffff", "#0000ff", "#0000ff"),min=-0.25,max=0.75)

#-----
# Criando imagem NDMI
#-----
getNDMI <- function(image) {
  return(image$normalizedDifference(c("B8", "B11")))
}
ndmi <- getNDMI(img)
ndmiPar <- list(palette = c(
  "#d73027", "#f46d43", "#fdad61", "#fee08b",
  "#d9ef8b", "#a6d96a", "#66bd63", "#1a9850"
),min=-1,max=1)
```

E criamos nosso mapa com os três índices usando:

```
#-----
# Centrando Mapa e Adicionando Imagens com parâmetros
# criados acima
#-----
Map$setCenter(-44.366,-17.69, zoom = 10)
Map$addLayer(ndvi1, ndviPar, "NDVI") + Map$addLayer(ndwi, ndwiPar,
"NDWI") + Map$addLayer(ndmi, ndmiPar, "NDMI")
```

O resultado é a criação de um mapa com as três imagens dos índices (NDVI, NDWI e NDMI).

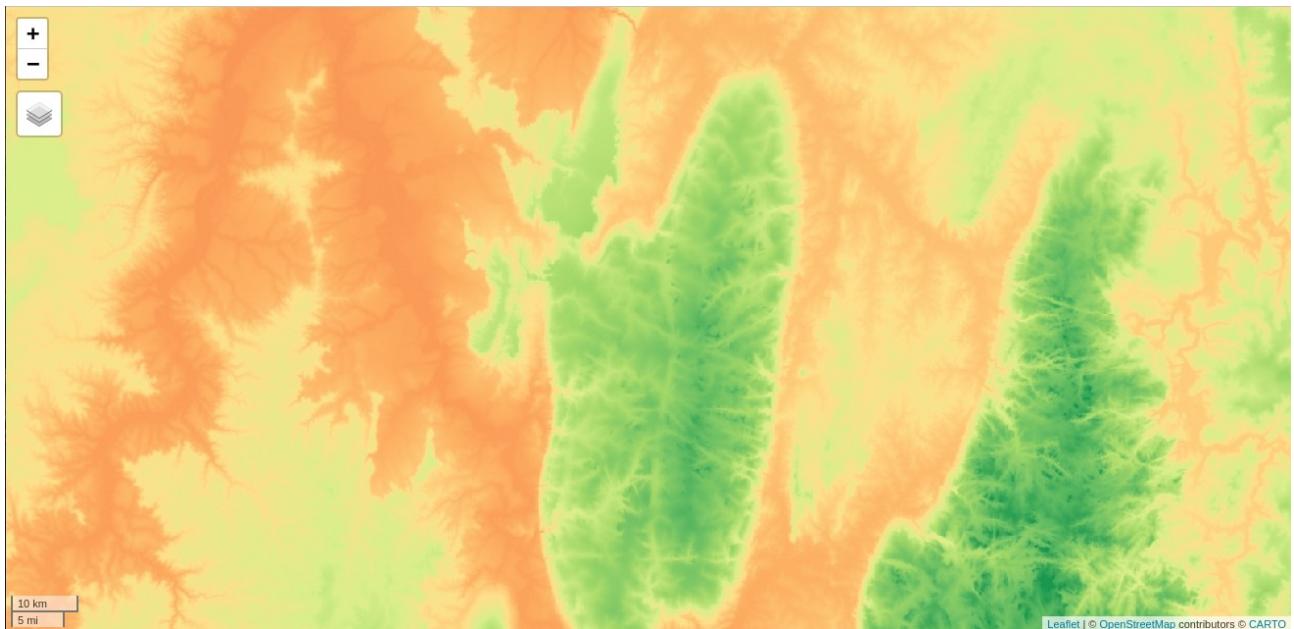


A grande vantagem de usar o rgee é que, além da vasta opção de imagens, as funções já são pré estabelecidas do tipo: `image$normalizedDifference(c(banda, banda))` para o cálculo de índices normalizados e não precisamos nos preocupar com diferente resoluções entre bandas como B5(20m) e B3(10m) no NDWI, pois elas são ajustadas automaticamente pelo GEE.

Um outro exemplo usando agora um DEM ALOS AW3D30 e extrairemos o gradiente e a direção(aspecto).

```
library(rgee)
ee_Initialize()
dado <- ee$Image ("JAXA/ALOS/AW3D30_V1_1")
elevação <- dado$select ("AVE")
Map$setCenter (-44.366, -17.69, zoom = 10)
Map$addLayer(eeObject=elevação, visParams=list(palette = c(
  "#d73027", "#f46d43", "#fdbe61", "#fee08b", "#d9ef8b", "#a6d96a",
  "#66bd63", "#1a9850"), min=200, max=1400), name="Elevação")
```

O resultado é:



RGEE possui muita versatilidade e vamos ver aqui mais alguns exemplos interessantes antes de nos aprofundarmos vendo em `ee$Image` e `ee$ImageCollection` nos próximos capítulos.

DADOS DE OUTRAS FONTES

Podemos mesclar dados de fontes distintas em um único mapa e isso pode ser feito de forma bem direta, carregar dados e uma fonte e em seguida carregamos dados de outra fonte a finalmente adicionamos em um único mapa.

Vamos carregar uma imagem Aster com as bandas B3N, B02 e B01 e em seguida vamos criar um hillshade para sobrepor a esta imagem usando transparência. Isso irá criar uma sensação de relevo.

```
library(rgee)
ee_Initialize()
# rotina de carregamento de imagem Aster
aster<-ee$ImageCollection("ASTER/AST_L1T_003")
ponto <- ee$Geometry$Point(-44.21722,-18.31933)
filtro<-aster$filterBounds(ponto)$filterDate('2013-06-15', '2013-09-15')

#selecionando as bandas
img <- filtro$select(c('B3N','B02','B01'))
vPar <- list(min = 0,max = 255)

# transformando graus em radianos
Radians <- function(img) {
  img$toFloat()$multiply(base::pi)$divide(180)
}
# Função que gera o hillshade
Hillshade <- function(az, ze, slope, aspect) {
  azimuth <- Radians(ee$image(az))
  zenith <- Radians(ee$image(ze))
  azimuth$subtract(aspect)$cos()$multiply(slope$sin())$multiply(zenith$sin())$add(zenith$cos())$multiply(slope$cos()))
}

# Rotina de carregamento de dem ALOS
terreno <- ee$Algorithms$Terrain(ee$image("JAXA/ALOS/AW3D30_V1_1")$select("AVE"))
# extraindo slope e aspect
slope_img <- Radians(terreno$select("slope"))
aspect_img <- Radians(terreno$select("aspect"))
# coordenada central do mapa e nível de zoom
Map$setCenter(-44.21722,-18.31933, zoom = 11)
# Adicionando as duas imagem sendo hillshade com somente 25%
# opacidade
Map$addLayer(img$median(), vPar, "Imagen Aster")+
  Map$addLayer(Hillshade(135, -60, slope_img,
    aspect_img), visParams=list(opacity=0.25), 'Hillshade')
```

Imagen com hillshade (sombra de relevo)

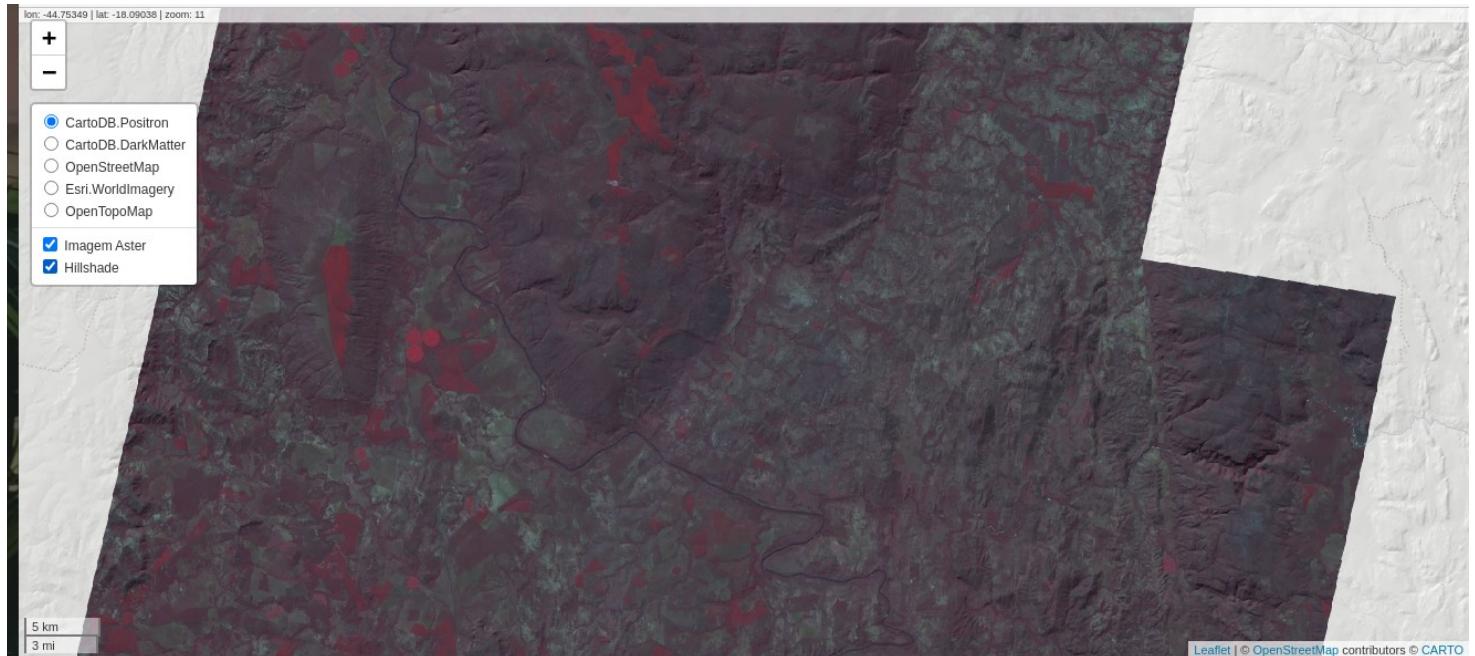
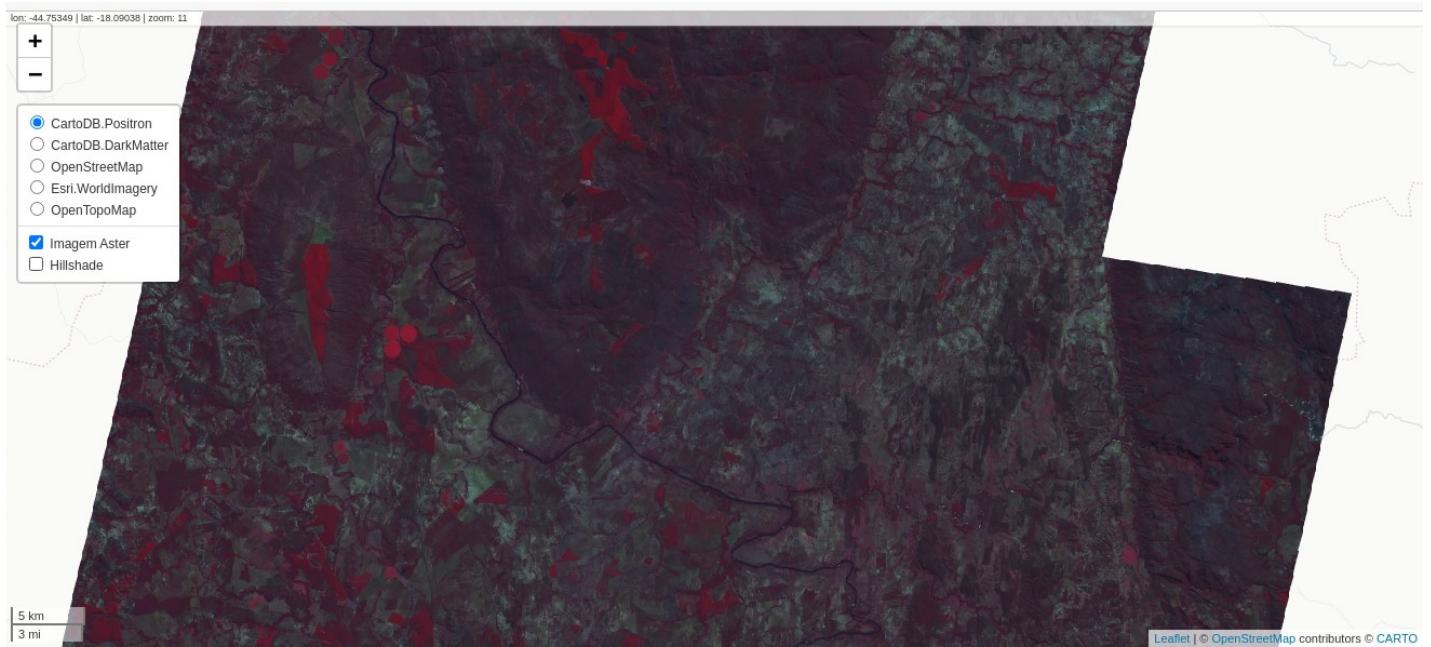
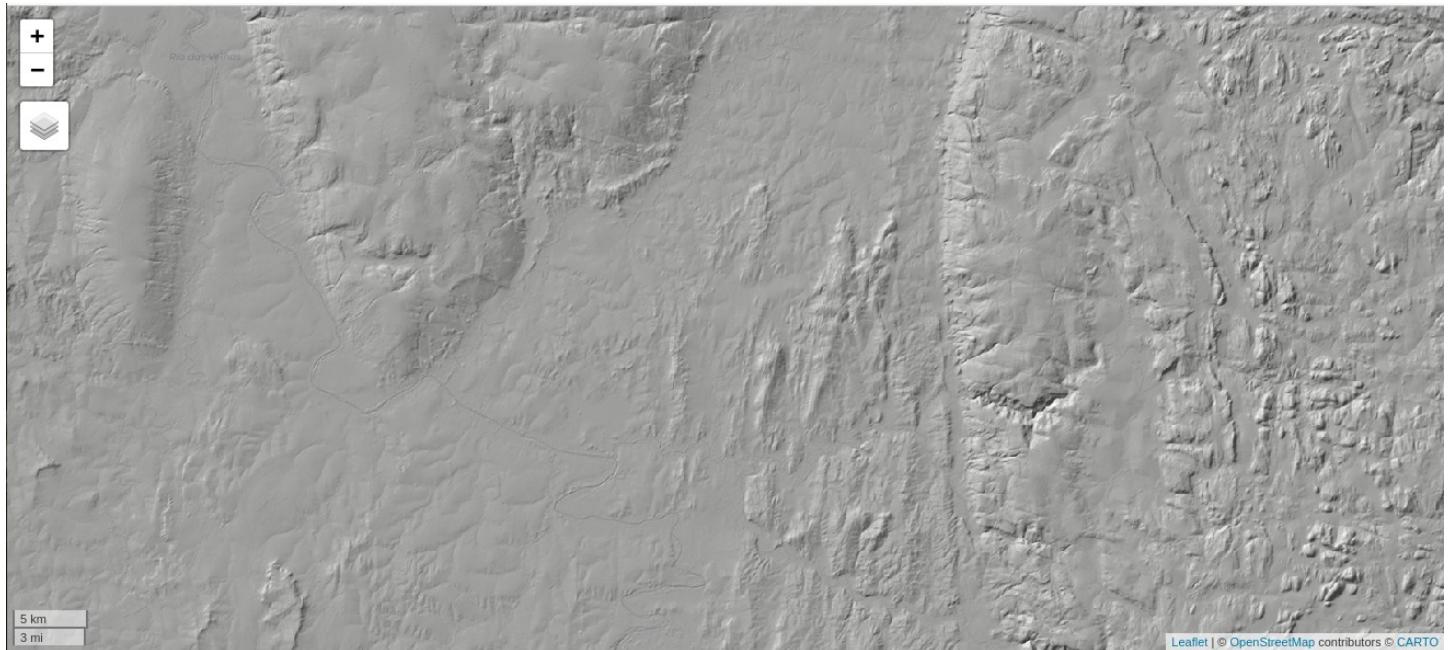


Imagen sem Relevo



podemos plotar somente o hillshade com pouca transparência usando:

```
Map$addLayer(Hillshade(135, -60, slope_img, aspect_img), visParams= list(opacity=0.75), 'Hillshade')
```



Vamos agora ver como carregar uma imagem Sentinel-1 e construir uma imagem composta com polarização e características de retroespalhamento.

```
library(rgee)
ee_Initialize()

# Carregando Sentinel-1 no ImageCollection.
sentinel1 <- ee$ImageCollection("COPERNICUS/S1_GRD")
$filterBounds(ee$Geometry$Point(-122.37383, 37.6193))

# Filtrando usando propriedades do metadado.
vh <-
sentinel1$filter(ee$Filter$listContains("transmitterReceiverPolarisation", "VV"))
$filter(ee$Filter$listContains("transmitterReceiverPolarisation", "VH"))$filter(ee$Filter$eq("instrumentMode", "IW"))

# Filtrando para obter imagens de diferentes ângulo de observação.
vhAscending <- vh$filter(ee$Filter$eq("orbitProperties_pass", "ASCENDING"))
vhDescending <- vh$filter(ee$Filter$eq("orbitProperties_pass", "DESCENDING"))

# Criando imagem composta com diferente polarização e ângulo de observação.
VH_Ascending_mean <- vhAscending$select("VH")$mean()
VV_Ascending_Descending_mean <- vhAscending$select("VV") %>%
ee$ImageCollection$merge(vhDescending$select("VV")) %>%
ee$ImageCollection$mean()
VH_Descending_mean <- vhDescending$select("VH")$mean()
composite <- ee$Image$cat(list(
```

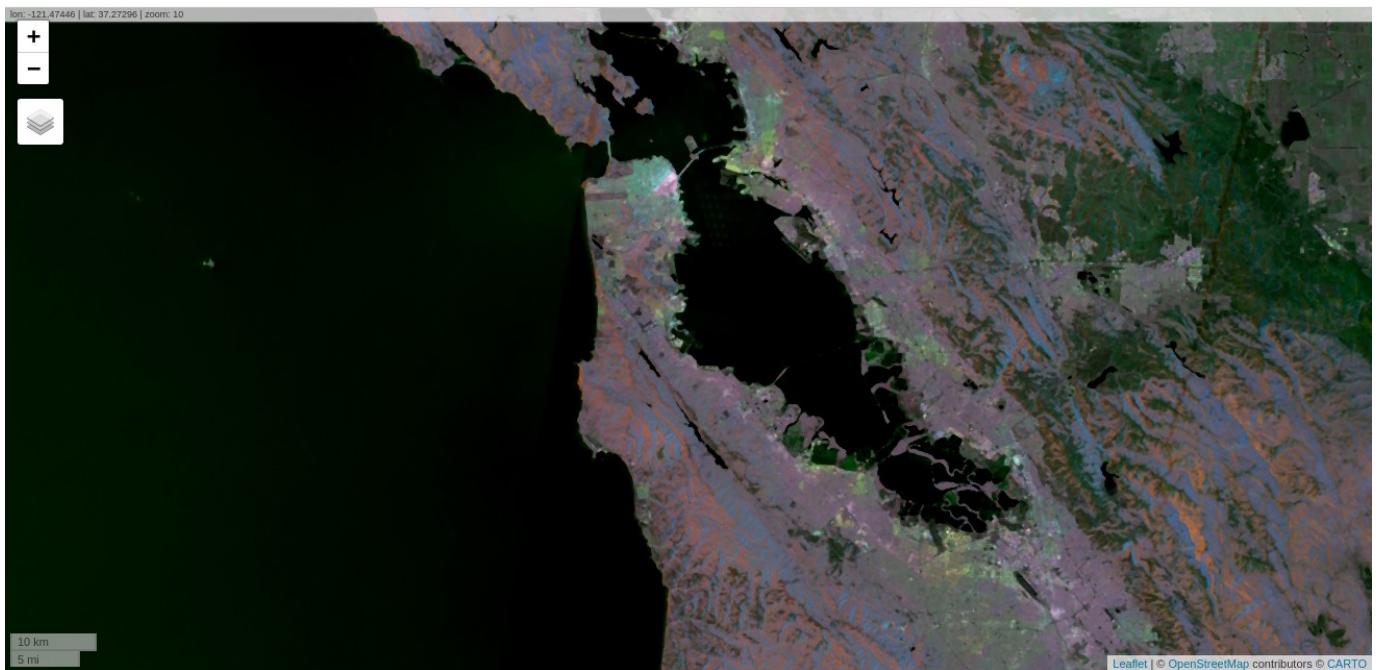
```

VH_Ascending_mean,
VV_Ascending_Descending_mean,
VH_Descending_mean
))$focal_median()

# Plotando como uma imagem composta de polarização e
# características de retroespalhamento.
Map$setCenter(-122.37383, 37.6193, 10)
Map$addLayer(
  composite,
  list(min = c(-25, -20, -25), max = c(0, 10, 0)),
  "composite"
)

```

O resultado é



Para concluirmos está parte introdutória, vamos ver como baixar dados do Google Earth Engine. Existe uma limitação de tamanho de imagem que pode ser baixada e também o caminho (path) gerado é temporário.

```

library(rgee)
ee_Initialize()
# carregando a imagem a ser baixada
image1 <- ee$ImageCollection('ASTER/AST_L1T_003')
$filterDate('2013-06-15', '2013-09-15')$median()
geom_params <- list(
  scale = 30,
  crs = 'EPSG:4326',
  region = '[[[-44.15, -18.25], [-44.35, -18.25], [-44.35, -18.35],
  [-44.15, -18.35]]]'
)
path <- image1$getDownloadUrl(geom_params)
print(path)

```

O path mostrado é uma url que será usada para baixar a imagem

```
[1]
"https://earthengine.googleapis.com/v1alpha/projects/earthengine-
legacy/thumbnails/132ba605f16c0350a77b608984414295-
3dafacadd346ad41d9cf658a765fd1f:getPixels"
```

Use wget na linha de comando (monitor) para baixar a imagem compactada e use unzip para extrair.

```
wget
https://earthengine.googleapis.com/v1alpha/projects/earthengine-
legacy/thumbnails/132ba605f16c0350a77b608984414295-
3dafacadd346ad41d9cf658a765fd1f:getPixels

unzip -x "132ba605f16c0350a77b608984414295-
3dafacadd346ad41d9cf658a765fd1f:getPixels"
```

De volta pro ambiente R use o pacote raster como é o normal para trabalhar com imagens locais. Vamos criar uma imagem termal e uma no campo do visível a partir dos dados baixados:

```
library(raster)
termal<-
stack('download.B10.tif','download.B11.tif','download.B12.tif')
plotRGB(termal,stretch='lin')
visivel<-
stack('download.B3N.tif','download.B02.tif','download.B01.tif')
plotRGB(visivel,stretch='lin')
```

A imagem termal (resolução 90m):



A imagem no campo do visível (resolução 15m):



UMA NOTA SOBRE COMO O GOOGLE EARTH ENGINE FUNCIONA

Quase toda transação no google earth engine é basicamente uma preparação de um dado JSON que num momento final é enviado ao servidor google earth engine e traduzido em uma imagem final num mapa ou em dados obtidos através de `getinfo()`.

Veja abaixo um exemplo que ilustra isso:

```
library(rgee)
ee_Initialize()
— rgee 0.6.1 ————— earthengine-api 0.1.225 —
  email: not_defined
  Initializing Google Earth Engine: DONE!
  Earth Engine user: users/???????????
```

```
aster<-ee$ImageCollection("ASTER/AST_L1T_003")
aster
ee.ImageCollection({
  "type": "Invocation",
  "arguments": {
    "id": "ASTER/AST_L1T_003"
  },
  "functionName": "ImageCollection.load"
})
```

```

ponto <- ee$Geometry$Point(-44.21722, -18.31933)
ponto
ee.Geometry({
  "type": "Point",
  "coordinates": [
    -44.21722,
    -18.31933
  ]
})
filtro<-aster$filterBounds(ponto)$filterDate('2013-06-15', '2013-09-15')
filtro
ee.ImageCollection({
  "type": "Invocation",
  "arguments": {
    "collection": {
      "type": "Invocation",
      "arguments": {
        "collection": {
          "type": "Invocation",
          "arguments": {
            "id": "ASTER/AST_L1T_003"
          },
          "functionName": "ImageCollection.load"
        },
        "filter": {
          "type": "Invocation",
          "arguments": {
            "leftField": ".all",
            "rightValue": {
              "type": "Invocation",
              "arguments": {
                "geometry": {
                  "type": "Point",
                  "coordinates": [
                    -44.21722,
                    -18.31933
                  ]
                }
              },
              "functionName": "Feature"
            }
          },
          "functionName": "Filter.intersects"
        }
      },
      "functionName": "Collection.filter"
    },
    "filter": {
      "type": "Invocation",
      "arguments": {
        "rightField": "system:time_start",
        "leftValue": {
          "type": "Invocation",
          "arguments": {
            "start": "2013-06-15",
            "end": "2013-09-15"
          },
          "functionName": "DateRange"
        }
      }
    }
}

```

```

        },
        "functionName": "Filter.dateRangeContains"
    }
},
"functionName": "Collection.filter"
)
bandas <- filtro$select(c('B3N', 'B02', 'B01'))
bandas
ee.ImageCollection({
  "type": "Invocation",
  "arguments": {
    "collection": {
      "type": "Invocation",
      "arguments": {
        "collection": {
          "type": "Invocation",
          "arguments": {
            "collection": {
              "type": "Invocation",
              "arguments": {
                "id": "ASTER/AST_L1T_003"
              },
              "functionName": "ImageCollection.load"
            },
            "filter": {
              "type": "Invocation",
              "arguments": {
                "leftField": ".all",
                "rightValue": {
                  "type": "Invocation",
                  "arguments": {
                    "geometry": {
                      "type": "Point",
                      "coordinates": [
                        -44.21722,
                        -18.31933
                      ]
                    }
                  },
                  "functionName": "Feature"
                }
              }
            },
            "functionName": "Filter.intersects"
          }
        }
      }
    }
  }
},
"functionName": "Collection.filter"
},
"filter": {
  "type": "Invocation",
  "arguments": {
    "rightField": "system:time_start",
    "leftValue": {
      "type": "Invocation",
      "arguments": {
        "start": "2013-06-15",
        "end": "2013-09-15"
      }
    },
    "functionName": "DateRange"
  }
}
,
```

```

        "functionName": "Filter.dateRangeContains"
    }
},
"functionName": "Collection.filter"
},
"baseAlgorithm": {
    "type": "Function",
    "argumentNames": [
        "_MAPPING_VAR_0_0"
    ],
    "body": {
        "type": "Invocation",
        "arguments": {
            "input": {
                "type": "ArgumentRef",
                "value": "_MAPPING_VAR_0_0"
            },
            "bandSelectors": [
                "B3N",
                "B02",
                "B01"
            ]
        },
        "functionName": "Image.select"
    }
},
"functionName": "Collection.map"
})
img <- bandas$median()
img
ee.Image({
    "type": "Invocation",
    "arguments": {
        "collection": {
            "type": "Invocation",
            "arguments": {
                "collection": {
                    "type": "Invocation",
                    "arguments": {
                        "collection": {
                            "type": "Invocation",
                            "arguments": {
                                "collection": {
                                    "type": "Invocation",
                                    "arguments": {
                                        "id": "ASTER/AST_L1T_003"
                                    },
                                    "functionName": "ImageCollection.load"
                                }
                            }
                        }
                    }
                }
            }
        }
    }
},
"filter": {
    "type": "Invocation",
    "arguments": {
        "leftField": ".all",
        "rightValue": {
            "type": "Invocation",
            "arguments": {
                "geometry": {
                    "type": "Point",
                    "coordinates": [

```

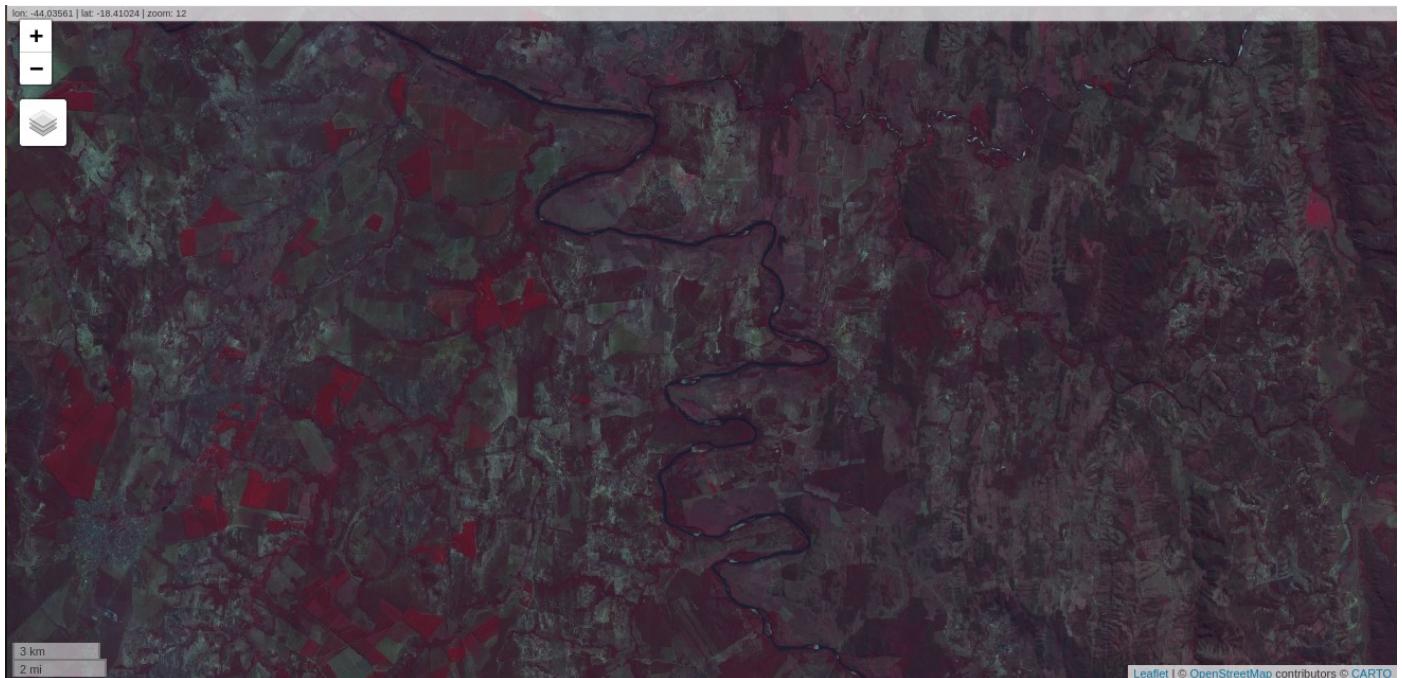
```

                -44.21722,
                -18.31933
            ]
        }
    },
    "functionName": "Feature"
}
},
"functionName": "Filter.intersects"
}
},
"functionName": "Collection.filter"
},
"filter": {
    "type": "Invocation",
    "arguments": {
        "rightField": "system:time_start",
        "leftValue": {
            "type": "Invocation",
            "arguments": {
                "start": "2013-06-15",
                "end": "2013-09-15"
            },
            "functionName": "DateRange"
        }
    },
    "functionName": "Filter.dateRangeContains"
}
},
"functionName": "Collection.filter"
},
"baseAlgorithm": {
    "type": "Function",
    "argumentNames": [
        "_MAPPING_VAR_0_0"
    ],
    "body": {
        "type": "Invocation",
        "arguments": {
            "input": {
                "type": "ArgumentRef",
                "value": "_MAPPING_VAR_0_0"
            },
            "bandSelectors": [
                "B3N",
                "B02",
                "B01"
            ]
        },
        "functionName": "Image.select"
    }
}
},
"functionName": "Collection.map"
}
},
"functionName": "reduce.median"
})
}
```

E agora centramos o mapa e plotamos.

```
Map$setCenter(-44.21722,-18.31933, zoom = 12)
Map$addLayer(img, visParams=list(min=0,max=255), "Imagen Aster")
```

Isso resultará no seguinte mapa:



Com o seguinte html (Note onde vem o mapa solicitado em negrito no código fonte da página):

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8"/>
<style>body{background-color:white;}</style>
...
<script>
<script src="lib/clipboard-0.0.1/setClipboardText.js"></script>

</head>
<body>
<div id="htmlwidget_container">
  <div id="htmlwidget-56d53ad4c5aa44506bdd" style="width:100%;height:400px;" class="leaflet html-widget"></div>
</div>
<script type="application/json" data-for="htmlwidget-56d53ad4c5aa44506bdd">{"x": {"options": {"minZoom": 1, "maxZoom": 52, "crs": {"crsClass": "L.CRS_EPSG3857", "code": null, "proj4def": null, "projectedBounds": null, "options": {}}, "preferCanvas": false, "bounceAtZoomLimits": false, "maxBounds": [[[[-90, -370], [90, 370]]]}}, "calls": [{"method": "addProviderTiles", "args": [{"CartoDB.Positron": 1, "CartoDB.Positron": {"errorTileUrl": "", "noWrap": false, "detectRetina": false}}, {"method": "addProviderTiles", "args": [{"CartoDB.DarkMatter": 2, "CartoDB.DarkMatter": {"errorTileUrl": "", "noWrap": false, "detectRetina": false}}]}, {"method": "addProviderTiles", "args": [{"OpenStreetMap": 3, "OpenStreetMap": {"errorTileUrl": "", "noWrap": false, "detectRetina": false}}]}, {"method": "addProviderTiles", "args": [{"Esri.WorldImagery": 4, "Esri.WorldImagery": {"errorTileUrl": "", "noWrap": false, "detectRetina": false}}]}]}
```

```

{"errorTileUrl":""}, "noWrap":false, "detectRetina":false}]]},  

{"method":"addProviderTiles", "args":["OpenTopoMap", 5, "OpenTopoMap",  

{"errorTileUrl":""}, "noWrap":false, "detectRetina":false}]],  

{"method":"addLayersControl", "args":  

[["CartoDB.Positron", "CartoDB.DarkMatter", "OpenStreetMap", "Esri.WorldImagery", "O  

penTopoMap"], [], {"collapsed":true, "autoZIndex":true, "position":"topleft"}]],  

{"method":"addScaleBar", "args":  

[{"maxWidth":100, "metric":true, "imperial":true, "updateWhenIdle":true, "position":  

"bottomleft"}]], {"method":"addTiles", "args":["https://  

earthengine.googleapis.com/v1alpha/projects/earthengine-legacy/maps/  

bf13ded46a01aef419158c6a870b572c-be9082f1e8f40fb31898a4fd75c18ad2/tiles/{z}/{x}/  

{y}", null, "Imagen Aster",  

{"minZoom":0, "maxZoom":18, "tileSize":256, "subdomains":"abc", "errorTileUrl":""}, "t  

ms":false, "noWrap":false, "zoomOffset":0, "zoomReverse":false, "opacity":1, "zIndex"  

:1, "detectRetina":false}], {"method":"addLayersControl", "args":  

[["CartoDB.Positron", "CartoDB.DarkMatter", "OpenStreetMap", "Esri.WorldImagery", "O  

penTopoMap"], "Imagen Aster",  

{"collapsed":true, "autoZIndex":true, "position":"topleft"}]],  

{"method":"hideGroup", "args": [null]}], "setView": [-18.31933, -44.21722], 11,  

[]], "evals":[], "jsHooks": {"render": [{"code": "function(el, x, data) {\n    return  

(\n        function(el, x, data) {\n            // get the leaflet map\n            var map =  

this; //HTMLWidgets.find('#' + el.id); \n            // we need a new div element  

because we have to handle\n            // the mouseover output separately\n            //  

debugger;\n            function addElement () {\n                // generate new div Element\n                var newDiv = $(document.createElement('div'));\n                // append at end of  

leaflet htmlwidget container\n                $(el).append(newDiv);\n                //provide ID  

and style\n                newDiv.addClass('lnlt');\n                newDiv.css({\n                    'position':  

'relative',\n                    'bottomleft': '0px',\n                    'background-color': 'rgba(255,  

255, 255, 0.7)',\n                    'box-shadow': '0 0 2px #bbb',\n                    'background-clip':  

'padding-box',\n                    'margin': '0',\n                    'padding-left': '5px',\n                    'color': '#333',\n                    'font': '9px/1.5 \"Helvetica Neue\", Arial, Helvetica,  

sans-serif',\n                    'z-index': '700',\n                });\n                return newDiv;\n            }\n        }\n    } // check for already existing lnlt class to not duplicate\n    var  

lnlt = $(el).find('.lnlt');\n    if(!lnlt.length) {\n        lnlt =  

addElement();\n        // grab the special div we generated in the beginning\n        // and put the mousmove output there\n        map.on('mousemove', function (e)  

{\n            if (e.originalEvent.ctrlKey) {\n                if  

(document.querySelector('.lnlt') === null) lnlt = addElement();\n                lnlt.text(\n                    'lon: ' + (e.latlng.lng).toFixed(5) + '\n                    ' | zoom: ' +  

map.getZoom() + '\n                    ' | x: ' +  

L.CRS.EPSG3857.project(e.latlng).x.toFixed(0) + '\n                    ' |  

y: ' + L.CRS.EPSG3857.project(e.latlng).y.toFixed(0) + '\n                    ' |  

epsg: 3857 + '\n                    ' | proj4: +proj=merc +a=6378137  

+b=6378137 +lat_ts=0.0 +lon_0=0.0 +x_0=0.0 +y_0=0 +k=1.0 +units=m  

+nadgrids=@null +no_defs ');\n            } else {\n                if  

(document.querySelector('.lnlt') === null) lnlt = addElement();\n                lnlt.text(\n                    'lon: ' + (e.latlng.lng).toFixed(5) + '\n                    ' | zoom: ' +  

map.getZoom() + ' ');\n            }\n        }\n    } // remove the lnlt div when  

mouse leaves map\n    map.on('mouseout', function (e) {\n        var strip =  

document.querySelector('.lnlt');\n        strip.remove();\n    });
    //$(el).keypress(67, function(e) {\n        map.on('preclick',  

function(e) {\n            if (e.originalEvent.ctrlKey) {\n                if  

(document.querySelector('.lnlt') === null) lnlt = addElement();\n                lnlt.text(\n                    'lon: ' + (e.latlng.lng).toFixed(5) + '\n                    ' | zoom: ' +  

map.getZoom() + ' ');\n            }\n        }
    });
    var txt =  

document.querySelector('.lnlt').textContent;\n    console.log(txt);\n    //txt.innerText.focus();\n    //txt.select();\n    setClipboardText('"' +  

txt + '"' );\n    ).call(this.getMap(), el, x,  

data);\n}, "data":null}])</script>
<script type="application/htmlwidget-sizing" data-for="htmlwidget-  

56d53ad4c5aa44506bdd">{ "viewer":  

{"width": "100%", "height": 400, "padding": 0, "fill": true}, "browser":  

{"width": "100%", "height": 400, "padding": 0, "fill": true}</script>  

</body>  

</html>
```

De forma simplística, toda a nossa instrução foi para preparar um dado JSON com as especificações do mapa que queremos gerar. Esse JSON é enviado para as entranhas do Google Earth Engine que preparará o mapa de acordo com sua receita e te envia uma URL temporária de tiles $\{z\}\{x\}\{y\}$

<https://earthengine.googleapis.com/v1alpha/projects/earthengine-legacy/maps/bf13ded46a01aef419158c6a870b572c-be9082f1e8f40fb31898a4fd75c18ad2/tiles/{z}/{x}/{y}>

para acessar a imagem e carregar no seu mapa dinamicamente (no nosso caso num mapa Leaflet).

Não precisamos nos preocupar com o lado "JSONês", o rgee existe para isso por ser vinculado ao python que, da mesma forma que o JS, são meros tradutores de instruções para o 'JSONês' a serem enviadas ao Google Earth Engine.

Vamos concentrar agora nos principais objetos e o que eles podem fazer. Iniciaremos pelo ee\$Image,

ee\$Image

Uma Imagem (objeto ee\$Image) é um dado raster que é representado como objeto Imagem no Earth Engine. Imagens são compostas por uma ou mais bandas e cada banda possui seu próprio nome, tipo de dado, escala, máscara e projeção. Cada imagem possui também metadados que contém um conjunto de propriedades desta.

O objeto imagem é o objetivo último em usar o Earth Engine e vamos começar por suas características e propriedades.

Construtores do ee\$Image

Imagens podem ser carregadas colocando o nome do ID de um dado que o Google Earth Engine possui no construtor ee\$Image. Você pode encontrar uma lista de ID no catálogo de dados no link <https://developers.google.com/earth-engine/datasets>.

Alguns conjuntos de dados (dataset) já geram um objeto do tipo ee\$Image outros do tipo ee\$ImageCollection.

Por exemplo, para carregar o dataset JAXA ALOS DEM do tipo ee\$Image usamos:

```
library(rgee)
ee_Initialize()
#Carregando a imagem DEM do globo todo
dem<-ee$Image("JAXA/ALOS/AW3D30_V1_1")
```

Usamos ee_print() para obter no formato texto as principais informações desta imagem

```
ee_print(dem,clean=TRUE)
```

Earth Engine Image —

Image Metadata:

- Class : ee\$Image
- ID : JAXA/ALOS/AW3D30_V1_1
- Number of Bands : 6
- Bands names : AVE MED AVE_STK MED_STK AVE_MSK MED_MSK
- Number of Properties : 25
- Number of Pixels* : 5.038848e+12
- Approximate size* : 2.20 TB

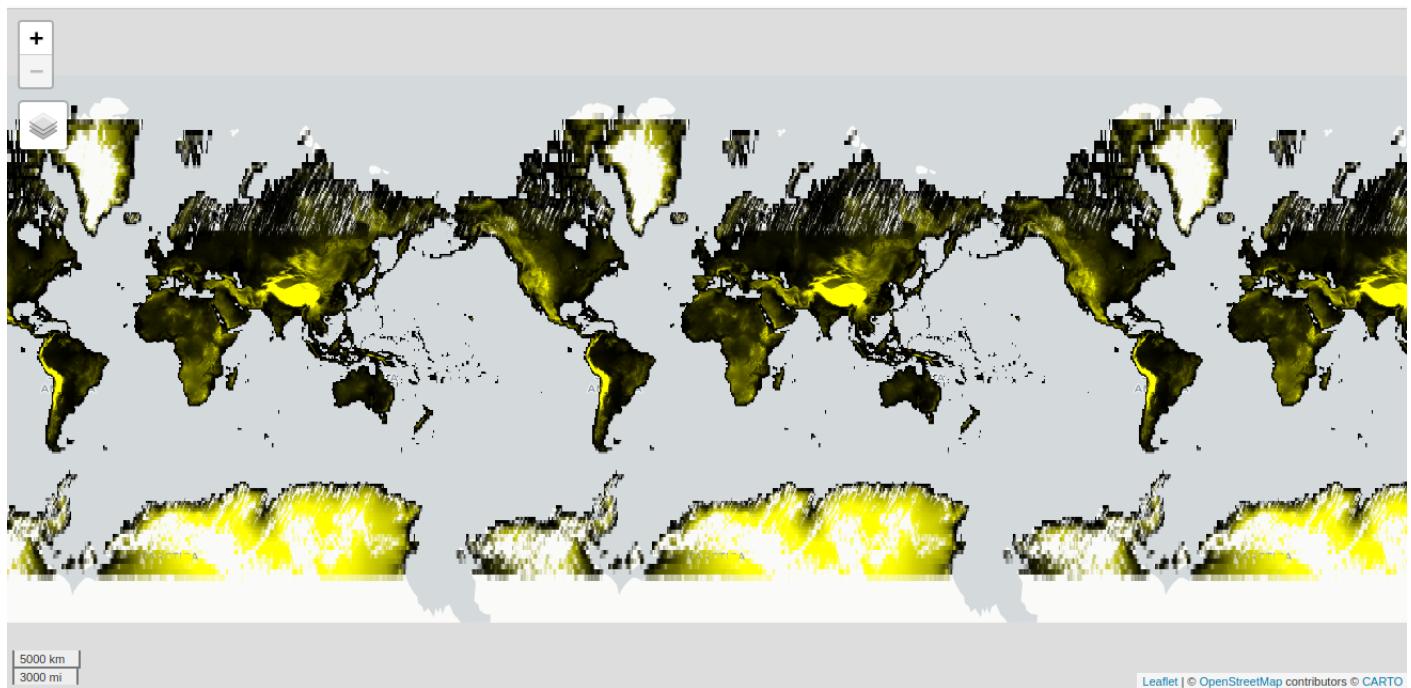
Band Metadata (img_band = AVE):

- EPSG (SRID) : 4326
- proj4string : +proj=longlat +datum=WGS84 +no_defs
- Geotransform : 0.0002777777778 0 -180 0 -0.0002777777778 83
- Nominal scale (meters) : 30.92208
- Dimensions : 1296000 648000
- Number of Pixels : 8.39808e+11
- Data type : INT
- Approximate size : 374.86 GB

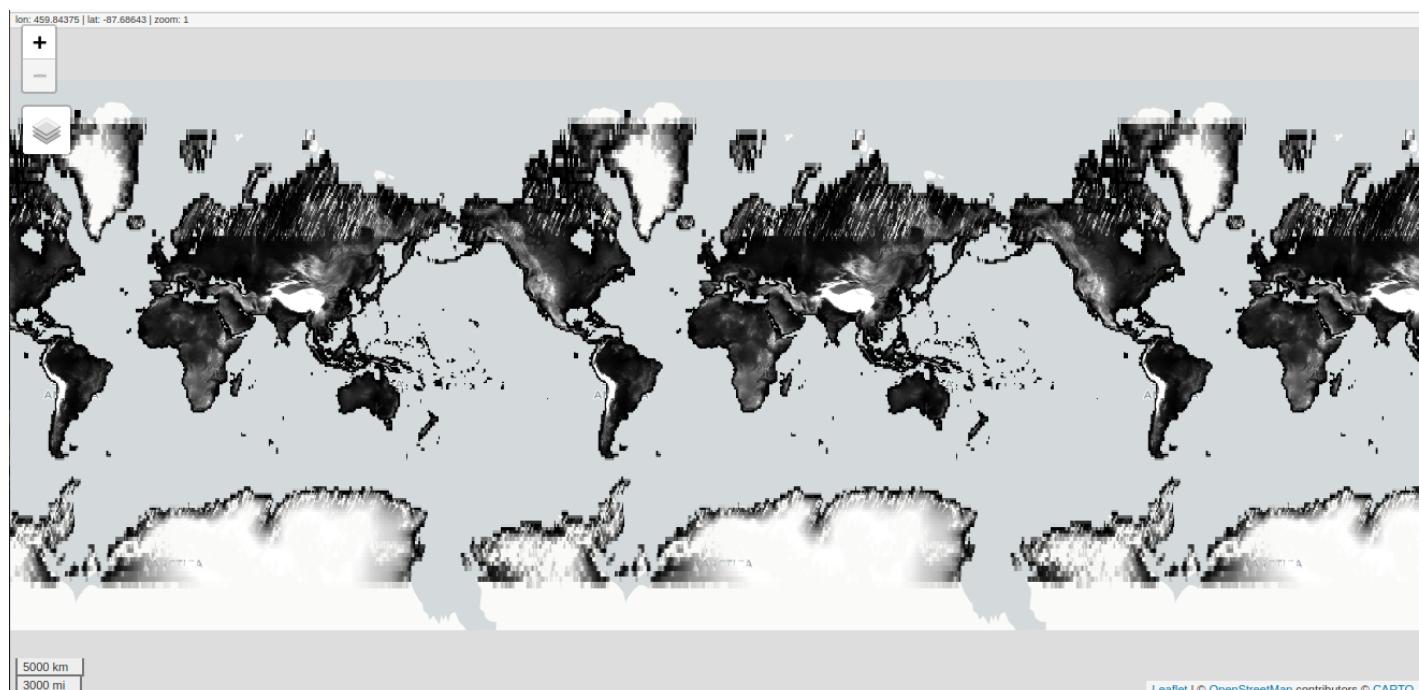
Ao plotarmos essa imagem veremos que se trata de inúmeros pixels com um valor. Mas se plotarmos a banda AVE ou MED veremos os valores de elevação associados a cada um dos pixels.

```
> Map$addLayer(dem, visParams=list(min=0,max=3000), "DEM")
> Map$addLayer(dem$select("AVE"),visParams=list(min=0,max=3000),
"DEM")
```

dem



dem\$select ("AVE")



Agora criaremos um objeto imagem a partir de uma ImageCollection do tipo Sentinel2.

```
library(rgee)
ee_Initialize()
# Selecionando a ee$ImageCollection
coleção<-ee$ImageCollection('COPERNICUS/S2_SR')
ponto <- ee$Geometry$Point(-44.366,-18.145)
início <- ee>Date("2019-07-11")
fim <- ee>Date("2019-07-20")
filtro<-coleção$filterBounds(ponto)$filterDate(início,fim)
#Aqui transformamos o ee#$ImageCollection em ee#$Image
img <- filtro$first()
```

Inspecionando os valores da imagem com ee_print().

```
ee_print(img,clear=TRUE)                                Earth Engine Image —
Image Metadata:
- Class          : ee$Image
- ID             : COPERNICUS/S2_SR/20190711T131251_20190711T131248_T23KNA
- Time start     : 2019-07-11 13:17:13
- Number of Bands : 23
- Bands names   : B1 B2 B3 B4 B5 B6 B7 B8 B8A B9 B11 B12 AOT WVP SCL
TCI_R TCI_G TCI_B MSK_CLDPRB MSK_SNWPRB QA10 QA20 QA60
- Number of Properties: 81
- Number of Pixels* : 77066790
- Approximate size* : 38.39 GB
Band Metadata (img_band = B1):
- EPSG (SRID)    : 32723
- proj4string     : +proj=utm +zone=23 +south +datum=WGS84 +units=m
+no_defs
- Geotransform   : 60 0 499980 0 -60 8100040
- Nominal scale(m) : 60
- Dimensions     : 1831 1830
- Number of Pixels : 3350730
- Data type      : INT
- Approximate size : 1.67 GB
```

Plotando o mapa como imagem RGB composta pelas bandas 4 ,3 e 2.

```
Map$setCenter(-44.366,-17.69, zoom = 10)
Map$addLayer(img,visParams=list(bands = c("B4", "B3", "B2"),min =
100,max = 8000,gamma = c(1.9,1.7,1.7)), "TCI")
```



Criando um objeto imagem a partir de uma nuvem de geotiff:

Você pode usar `ee$Image$loadGeoTIFF()` para criar imagem a partir do Geotiffs otimizados nas Nuvens no Google Cloud storage. Por exemplo, este dados Landsat hospedado no Google cloud contém esse geotiff composto pela bandas de uma cena do landsat8. Você pode carregar ela usando:

```
library(rgee)
ee_Initialize()
uri<-'gs://gcp-public-data-landsat/LC08/01/001/002/
LC08_L1GT_001002_20160817_20170322_01_T2/
LC08_L1GT_001002_20160817_20170322_01_T2_B5.TIF'
cloudImage<-ee$Image$loadGeoTIFF(uri)
```

Visualizando a informação da imagem usando `ee_print()`

```
ee_print(cloudImage, clean = TRUE) Earth Engine Image —
```

Image Metadata:

- Class	: ee\$Image
- ID	: no_id
- Number of Bands	: 1
- Bands names	: B0
- Number of Properties	: 0
- Number of Pixels*	: 80658261
- Approximate size*	: 123.07 MB

Band Metadata (img_band = B0):

- EPSG (SRID)	: 32630
- proj4string	: +proj=utm +zone=30 +datum=WGS84 +units=m
+no_defs	
- Geotransform	: 30 0 342000 0 -30 9016200
- Nominal scale (meters)	: 30
- Dimensions	: 8991 8971
- Number of Pixels	: 80658261
- Data type	: INT
- Approximate size	: 123.07 MB

E por último, veremos como criar e manipular um objeto imagem constante:

```
library(rgee)
ee_Initialize()
imagem<-ee$Image(1)
ee_print(imagem, clean = TRUE)

```

Image Metadata:

- Class	:	ee\$Image
- ID	:	no_id
- Number of Bands	:	1
- Bands names	:	constant
- Number of Properties	:	0
- Number of Pixels*	:	64800
- Approximate size*	:	101.25 KB

Band Metadata (img_band = constant):

- EPSG (SRID)	:	4326
- proj4string	:	+proj=longlat +datum=WGS84 +no_defs
- Geotransform	:	1 0 0 0 1 0
- Nominal scale (meters)	:	111319.5
- Dimensions	:	360 180
- Number of Pixels	:	64800
- Data type	:	INT
- Approximate size	:	101.25 KB

NOTE: (*) Properties calculated considering a constant geotransform and data type.

Podemos renomear a banda usando:

```
tresbandas <- ee$Image(c(1, 2, 3))
imagem2 <- tresbandas$select(
  opt_selectors = c("constant", "constant_1", "constant_2"),
  opt_names = c("banda1", "banda2", "banda3")
)
ee_print(imagem2)

```

Image Metadata:

- Class	:	ee\$Image
- ID	:	no_id
- Number of Bands	:	3
- Bands names	:	banda1 banda2 banda3
- Number of Properties	:	0
- Number of Pixels*	:	194400
- Approximate size*	:	303.75 KB

Band Metadata (img_band = banda1):

- EPSG (SRID)	:	4326
- proj4string	:	+proj=longlat +datum=WGS84 +no_defs
- Geotransform	:	1 0 0 0 1 0
- Nominal scale (meters)	:	111319.5
- Dimensions	:	360 180
- Number of Pixels	:	64800
- Data type	:	INT
- Approximate size	:	101.25 KB

Visualização de ee\$Image

Quando usamos Map\$addLayer o segundo parâmetro é uma lista de parâmetros a ser usados nessa imagem. Os itens dessa lista são:

bands : vetor de bandas que serão usadas nos canais RGB

min : número ou vetor de três números que serão os valores mínimos em termos percentuais usado(s) na(s) banda(s).

max : número ou vetor de três números que serão os valores máximos em termos percentuais usado(s) na(s) banda(s).

gain : número ou vetor de três números que serão os valores de ganho usado(s) na(s) banda(s).

bias : número ou vetor de três números que serão os valores de ganho adicionado(s) a cada DN (Digital Number ou pixel) da(s) banda(s).

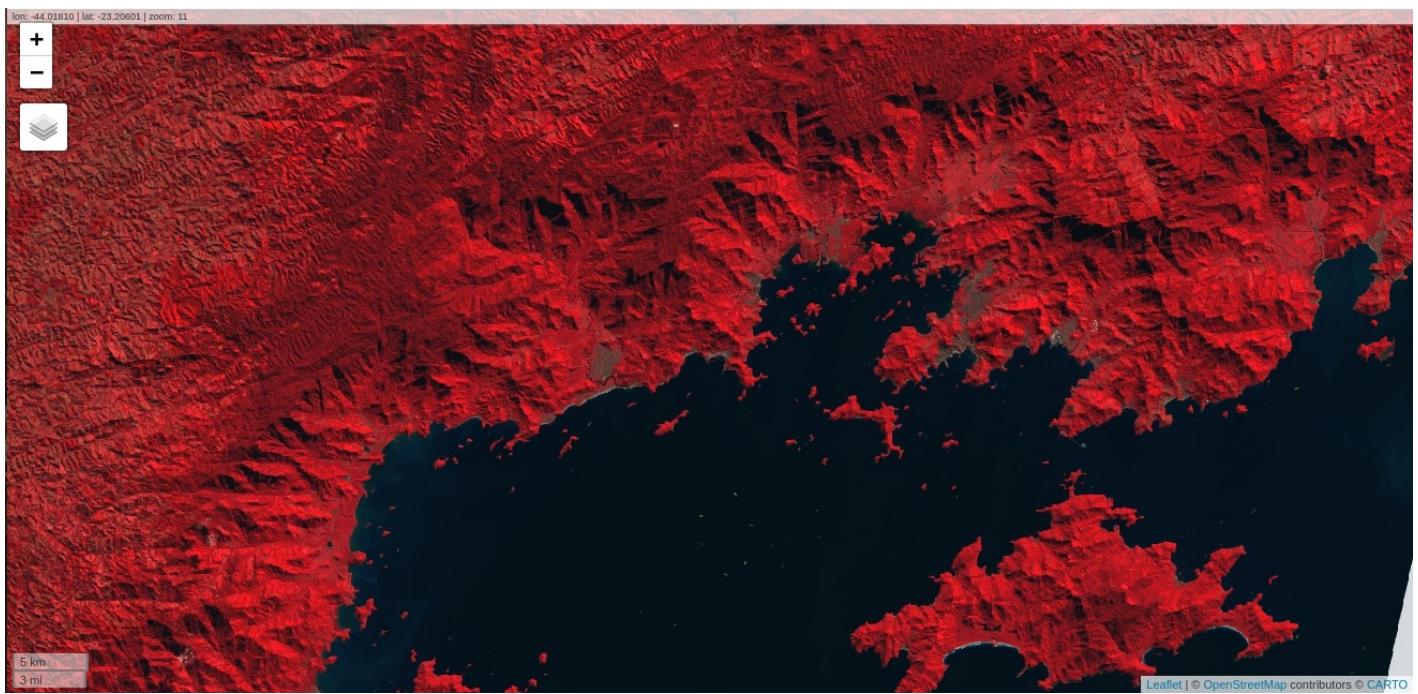
gamma : número ou vetor de três números que serão o(s) valor(es) de correção gama (luminosidade) da(s) banda(s).

palette : vetor de strings de cor no estilo CSS que será aplicado como rampa de cor da imagem. Só para imagens com uma única banda.

opacity : valor usado para ajustar a transparência da camada (0 totalmente transparente 1 totalmente opaco).

Vamos criar uma composição RGB com base nos parâmetros de visualização:

```
library(rgee)
ee_Initialize()
coleção<-ee$ImageCollection('LANDSAT/LC08/C01/T1_TOA')
ponto <- ee$Geometry$Point(-44.4678,-23.0058)
início <- ee>Date("2019-06-11")
fim <- ee>Date("2019-08-20")
filtro<-coleção$filterBounds(ponto)$filterDate(início,fim)
imagem <- filtro$filterFirst()
imgViz <- list(
  min = 0,
  max = 0.5,
  bands = c("B5", "B4", "B3"),
  gamma = c(0.95, 1.1, 1)
)
Map$setCenter(-44.4678,-23.0058, 10)
Map$addLayer(imagem, imgViz, 'Imagen Landsat 8')
```



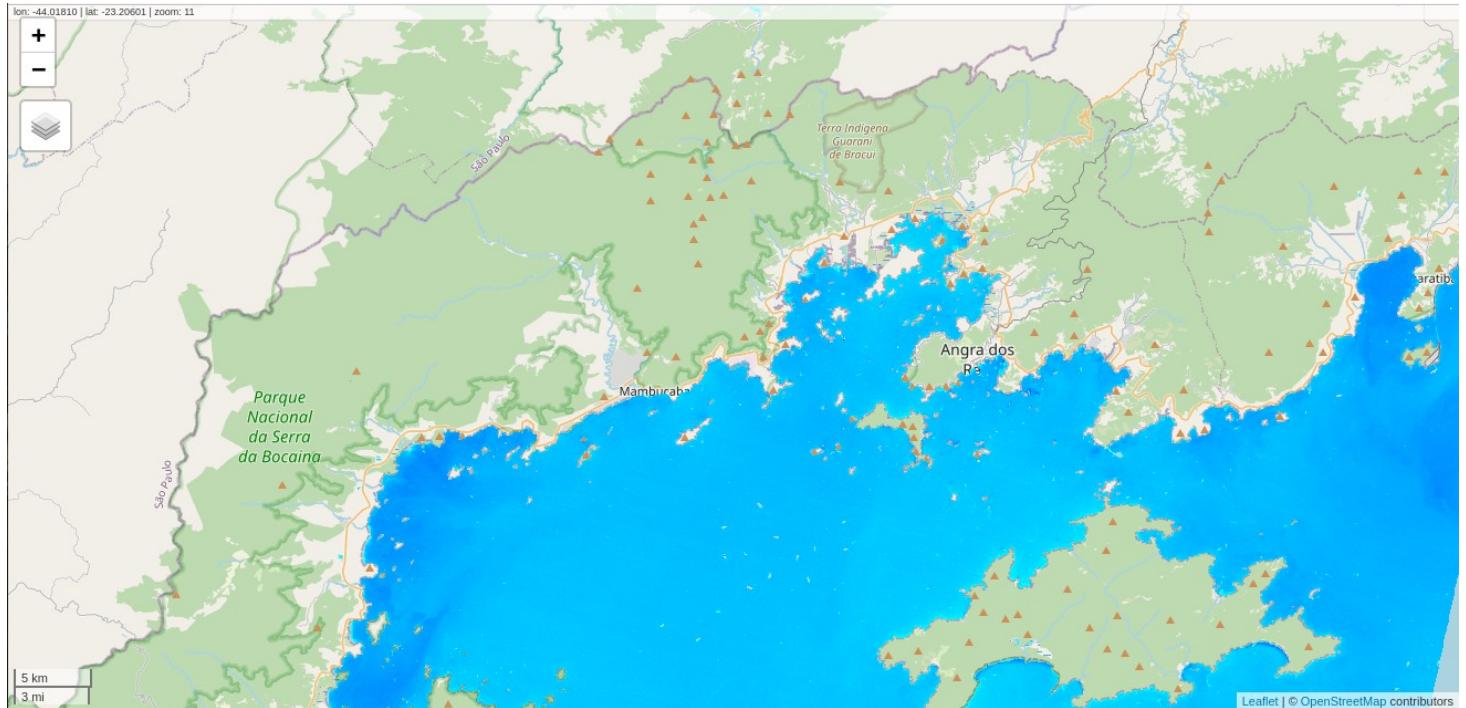
Vamos agora gerar uma imagem de uma banda da mesma cena. Vamos criar uma imagem NDWI usando paleta de cor.

```
#Criando um ndwi (separa a água)
ndwi <- imagem$normalizedDifference(c("B3", "B5"))
ndwiViz <- list(
  min = 0.5,
  max = 1,
  palette = c("00FFFF", '0080FF', "0000FF")
)
Map$addLayer(ndwi, ndwiViz, 'NDWI')
```



Vamos criar uma máscara usando `image$updateMask()` para tornar opacos valores válidos (não nulos) abaixo de um certo valor na imagem NDWI, nesse caso valores menores que 0.4, ou seja, parte que não é água se tornaram transparentes.

```
ndwiMascara <- ndwi$updateMask(ndwi$gte(0.4))  
Map$addLayer(ndwiMascara, ndwiViz, 'Máscara NDWI')
```

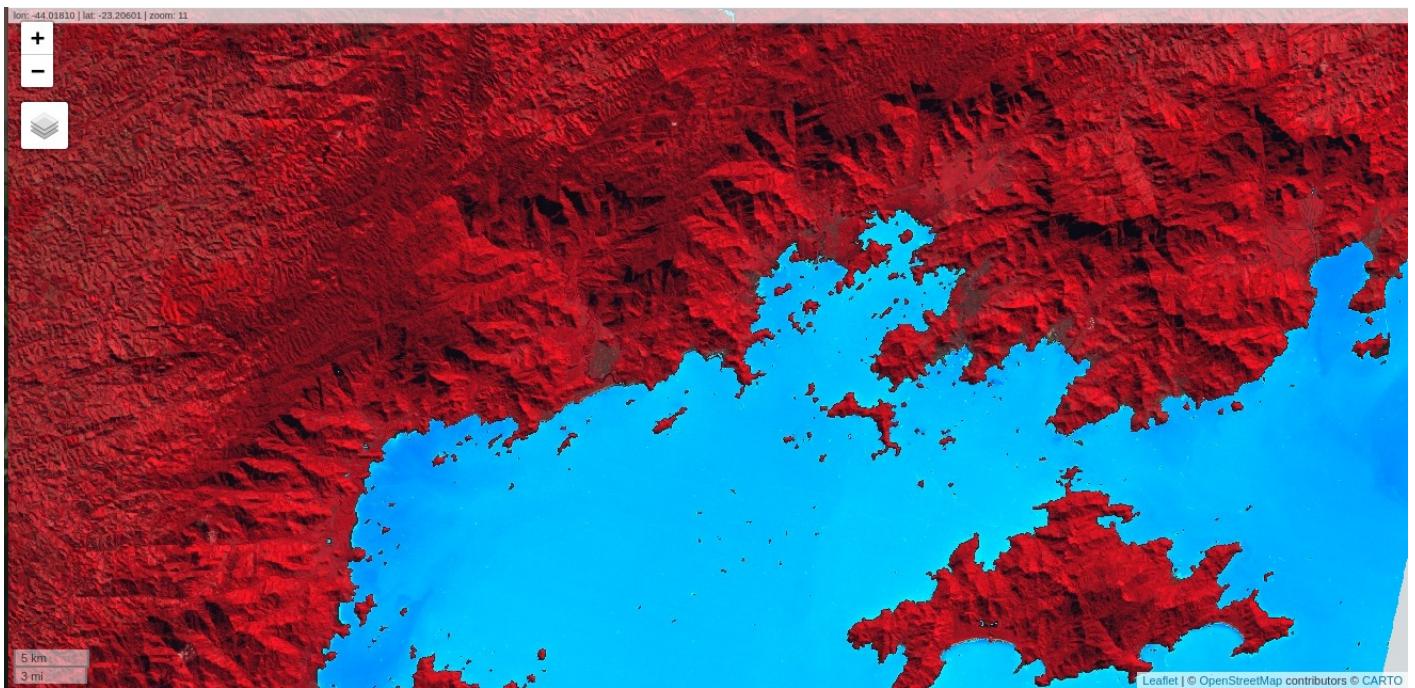


Podemos usar a função `image$visualize()` para criar uma imagem RGB de qualquer imagem.

```
imagemRGB<-imagem$visualize(bands=c('B5','B4','B3'),max=0.5)  
ndwiRGB<- ndwiMasked$visualize(min=0.5,max=1,palette= c('#00FFFF', '#0000FF'))
```

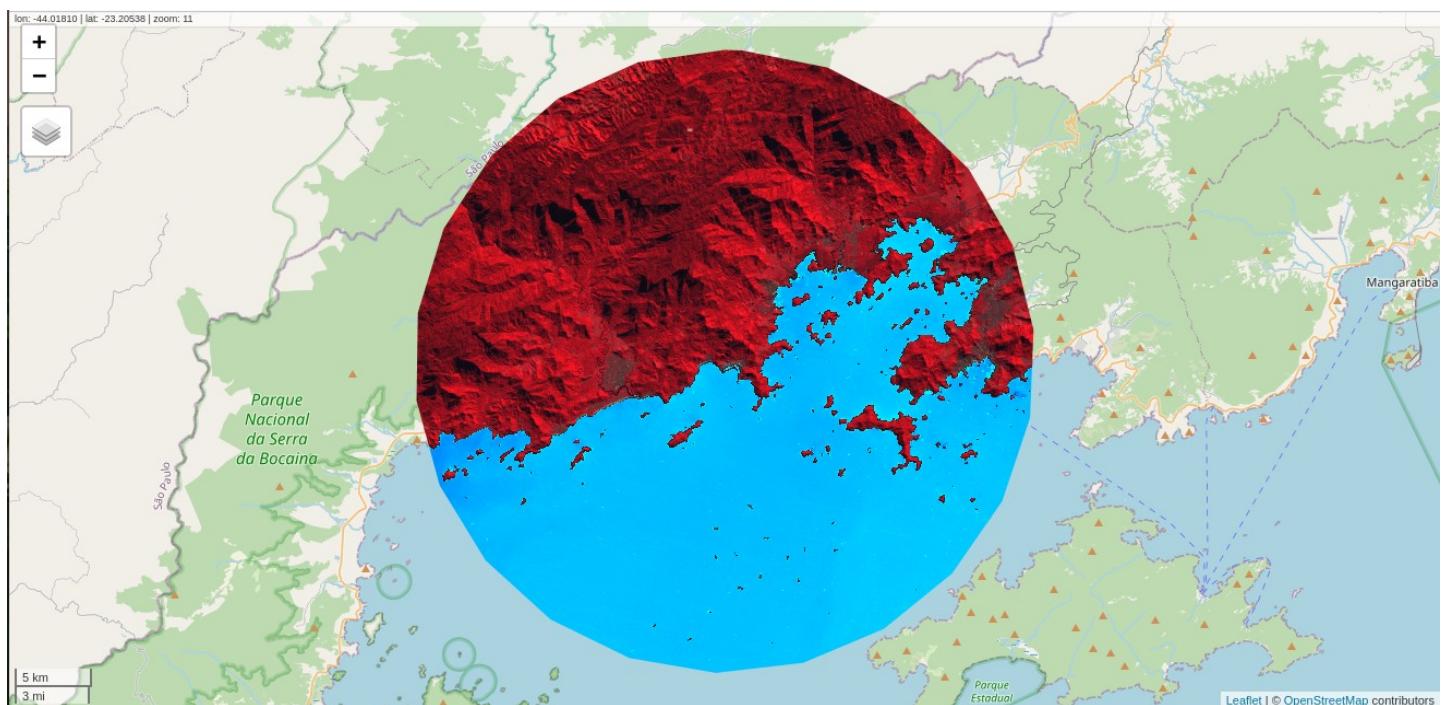
E podemos criar mosaicos destas duas imagens RGB usando:

```
mosaico<-ee$ImageCollection(c(imagemRGB, ndwiRGB))$mosaic();  
Map$addLayer(mosaico ,list(), 'mosaico')
```



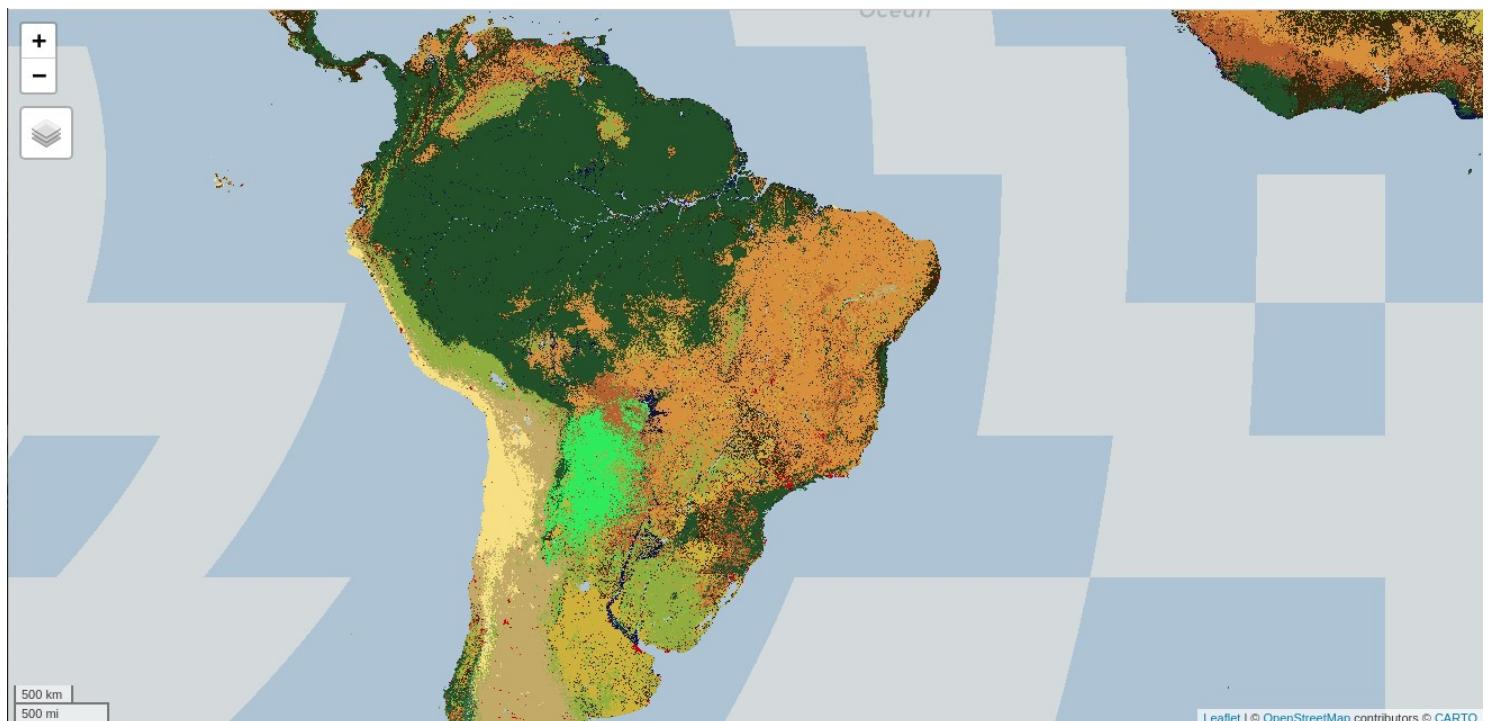
E podemos extrair uma região da imagem baseado em um objeto geométrico qualquer:

```
roi <- ee$Geometry$Point(c(-44.4678, -23.0058))$buffer(20000)
Map$addLayer(mosaico$clip(roi))
```



Agora vamos usar uma imagem MODIS para criar um "mapa de biomassas":

```
biomas<-ee$Image('MODIS/051/MCD12Q1/2012_01_01')
$select('Land_Cover_Type_1')
igbpPalette<-c('#aec3d4', '#152106', '#225129', '#369b47',
  '#30eb5b', '#387242', '#6a2325', '#c3aa69', '#b76031',
  '#d9903d', '#91af40', '#111149', '#cdb33b', '#cc0013',
  '#33280d', '#d7cdcc', '#f7e084', '#6f6f6f')
Map$setCenter(-50.0, -15.0, 4)
Map$addLayer(biomas, list(min=0, max=17, palette = igbpPalette),
  'Biomass')
```



Informações e metadata do ee\$image

Já vimos acima como obter algumas informações e metadata da imagem usando `ee_print`. Vamos agora ver como obter mais informações e metadata individualmente. Primeiro vamos carregar uma imagem e novamente usar `ee_print()`.

```
coleção<-ee$ImageCollection('COPERNICUS/S2_SR')
ponto <- ee$Geometry$Point(-44.366, -18.145)
início <- ee>Date("2019-07-11")
fim <- ee>Date("2019-07-20")
filtro<-coleção$filterBounds(ponto)$filterDate(início, fim)
img <- filtro$first()
ee_print(img)
```

Earth Engine Image —

Image Metadata:

- Class	:	ee\$image
---------	---	-----------

```

- ID : COPERNICUS/S2_SR/20190711T131251_20190711T131248_T23KNA
- Time start : 2019-07-11 13:17:13
- Number of Bands : 23
- Bands names : B1 B2 B3 B4 B5 B6 B7 B8 B8A B9 B11 B12 AOT WVP SCL
TCI_R TCI_G TCI_B MSK_CLDPRB MSK_SNWPRB QA10 QA20 QA60
- Number of Properties: 81
- Number of Pixels* : 77066790
- Approximate size* : 38.39 GB
Band Metadata (img_band = B1):
- EPSG (SRID) : 32723
- proj4string : +proj=utm +zone=23 +south +datum=WGS84 +units=m
+no_defs
- Geotransform : 60 0 499980 0 -60 8100040
- Nominal scale(m) : 60
- Dimensions : 1831 1830
- Number of Pixels : 3350730
- Data type : INT
- Approximate size : 1.67 GB

```

NOTE: (*) Properties calc. considering a constant geotransform and data type.

Nomes das bandas presentes na imagem

```

NomeBandas<-img$bandNames()
cat("Bandas: ",paste(NomeBandas$info(),"\n",collapse=" "))
Bandas: B1
B2
B3
B4
B5
B6
B7
B8
B8A
B9
B11
B12
AOT
WVP
SCL
TCI_R
TCI_G
TCI_B
MSK_CLDPRB
MSK_SNWPRB
QA10
QA20
QA60

```

Vendo as informações de projeção da banda 1:

```

b1proj<- img$select('B1')$projection()
cat("Projeção B1: ", paste(b1proj$info(),"\n", collapse = " "))
Projeção B1: Projection
EPSG:32723
c(60, 0, 499980, 0, -60, 8100040)

```

Escala da banda 1:

```

b1escala<-img$select('B1')$projection()$nominalScale()
cat("Escala B1: ", paste(b1escala$info(),"\n", collapse = " "))
Escala B1: 60

```

Escala da banda 2:

```
b2escala<-img$select('B2')$projection()$nominalScale()
cat("Escala B2: ", paste(b2escala getInfo(),"\n", collapse = " "))
Escala B2: 10
```

Escala da banda 5:

```
b5escala<-img$select('B5')$projection()$nominalScale()
cat("Escala B5: ", paste(b5escala getInfo(),"\n", collapse = " "))
Escala B5: 20
```

Mostrando todos os Metadados:

```
metadados<-img$propertyNames()
cat("Metadados: ", paste(metadados getInfo(),"\n", collapse = " "))
Metadados:
DATATAKE_IDENTIFIER
AOT_RETRIEVAL_ACCURACY
SPACECRAFT_NAME
SATURATED_DEFECTIVE_PIXEL_PERCENTAGE
system:id
MEAN_INCIDENCE_AZIMUTH_ANGLE_B8A
CLOUD_SHADOW_PERCENTAGE
MEAN_SOLAR_AZIMUTH_ANGLE
system:footprint
VEGETATION_PERCENTAGE
SOLAR_IRRADIANCE_B12
system:version
SOLAR_IRRADIANCE_B10
SENSOR_QUALITY
SOLAR_IRRADIANCE_B11
GENERATION_TIME
SOLAR_IRRADIANCE_B8A
FORMAT_CORRECTNESS
CLOUD_COVERAGE_ASSESSMENT
THIN_CIRRUS_PERCENTAGE
system:time_end
WATER_VAPOUR_RETRIEVAL_ACCURACY
system:time_start
DATASTRIP_ID
PROCESSING_BASELINE
SENSING_ORBIT_NUMBER
NODATA_PIXEL_PERCENTAGE
SENSING_ORBIT_DIRECTION
GENERAL_QUALITY
GRANULE_ID
REFLECTANCE_CONVERSION_CORRECTION
MEDIUM_PROBA_CLOUDS_PERCENTAGE
MEAN_INCIDENCE_AZIMUTH_ANGLE_B8
DATATAKE_TYPE
MEAN_INCIDENCE_AZIMUTH_ANGLE_B9
MEAN_INCIDENCE_AZIMUTH_ANGLE_B6
MEAN_INCIDENCE_AZIMUTH_ANGLE_B7
MEAN_INCIDENCE_AZIMUTH_ANGLE_B4
MEAN_INCIDENCE_ZENITH_ANGLE_B1
NOT_VEGETATED_PERCENTAGE
MEAN_INCIDENCE_AZIMUTH_ANGLE_B5
RADIOMETRIC_QUALITY
MEAN_INCIDENCE_AZIMUTH_ANGLE_B2
MEAN_INCIDENCE_AZIMUTH_ANGLE_B3
MEAN_INCIDENCE_ZENITH_ANGLE_B5
MEAN_INCIDENCE_AZIMUTH_ANGLE_B1
MEAN_INCIDENCE_ZENITH_ANGLE_B4
MEAN_INCIDENCE_ZENITH_ANGLE_B3
MEAN_INCIDENCE_ZENITH_ANGLE_B2
```

```
MEAN_INCIDENCE_ZENITH_ANGLE_B9
MEAN_INCIDENCE_ZENITH_ANGLE_B8
MEAN_INCIDENCE_ZENITH_ANGLE_B7
DARK_FEATURES_PERCENTAGE
HIGH_PROBA_CLOUDS_PERCENTAGE
MEAN_INCIDENCE_ZENITH_ANGLE_B6
UNCLASSIFIED_PERCENTAGE
MEAN_SOLAR_ZENITH_ANGLE
MEAN_INCIDENCE_ZENITH_ANGLE_B8A
RADIATIVE_TRANSFER_ACCURACY
MGRS_TILE
CLOUDY_PIXEL_PERCENTAGE
PRODUCT_ID
MEAN_INCIDENCE_ZENITH_ANGLE_B10
SOLAR_IRRADIANCE_B9
SNOW_ICE_PERCENTAGE
DEGRADED_MSI_DATA_PERCENTAGE
MEAN_INCIDENCE_ZENITH_ANGLE_B11
MEAN_INCIDENCE_ZENITH_ANGLE_B12
SOLAR_IRRADIANCE_B6
MEAN_INCIDENCE_AZIMUTH_ANGLE_B10
SOLAR_IRRADIANCE_B5
MEAN_INCIDENCE_AZIMUTH_ANGLE_B11
SOLAR_IRRADIANCE_B8
MEAN_INCIDENCE_AZIMUTH_ANGLE_B12
SOLAR_IRRADIANCE_B7
SOLAR_IRRADIANCE_B2
SOLAR_IRRADIANCE_B1
SOLAR_IRRADIANCE_B4
GEOMETRIC_QUALITY
SOLAR_IRRADIANCE_B3
system:asset_size
WATER_PERCENTAGE
system:index
system:bands
system:band_names
```

Obtendo o valor de um campo de metadados listado acima:

```
nuvens <- img$get('CLOUD_SHADOW_PERCENTAGE')
cat("Sombra de Nuvens:", paste(nuvens getInfo(), "\n", collapse=" "))

Sombra de Nuvens: 0.000116
```

Operações aritméticas com ee\$image

O google earth engine tem duas formas de executar operações aritméticas: uma esquisitona para usuários de R que usa funções em linha para cada operador que é ok para operações simples e outra usando a função ee\$image\$expression() que é mais apropriada.

Efetuando o NDVI entre duas bandas pelo método de funções em linha:

```
library(rgee)
ee_Initialize()
coleção<-ee$imageCollection('LANDSAT/LC08/C01/T1_TOA')
```

```

ponto <- ee$Geometry$Point (-44.4678, -23.0058)
início <- ee>Date ("2019-06-11")
fim <- ee>Date ("2019-08-20")
filtro<-coleção$filterBounds(ponto)$filterDate(início,fim)
imagem <- filtro$first()
ndvi<-(imagem$select('B4')$subtract(imagem$select('B3'))))
$divide(imagem$select('B4')$add(imagem$select('B3'))))

```

E agora calculando o EVI usando a função `expression()`:

```

evi<-imagem$expression(
  '2.5 * ((NIR - RED) / (NIR + 6 * RED - 7.5 * BLUE))',
  list ('NIR'=imagem$select('B5'),
    'RED'=imagem$select('B4'),
    'BLUE'=imagem$select('B2')))

```

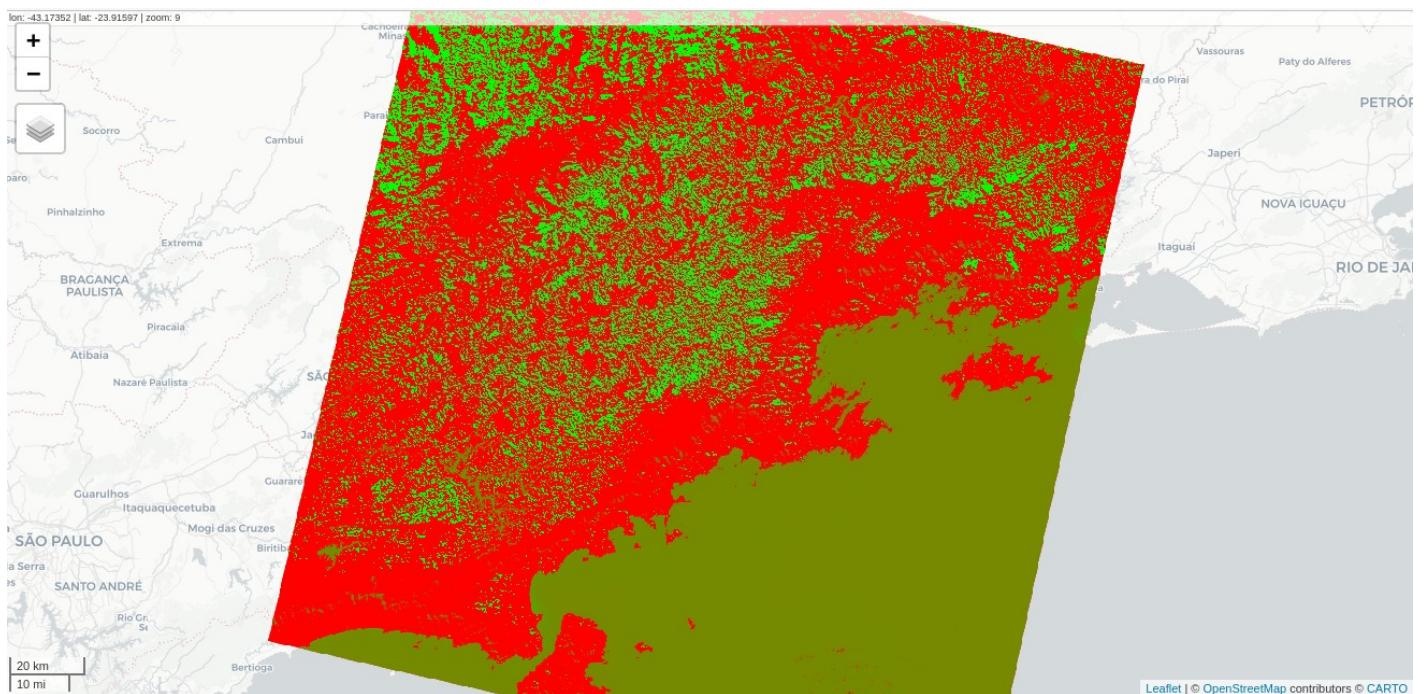
Plotando as duas imagens:

```

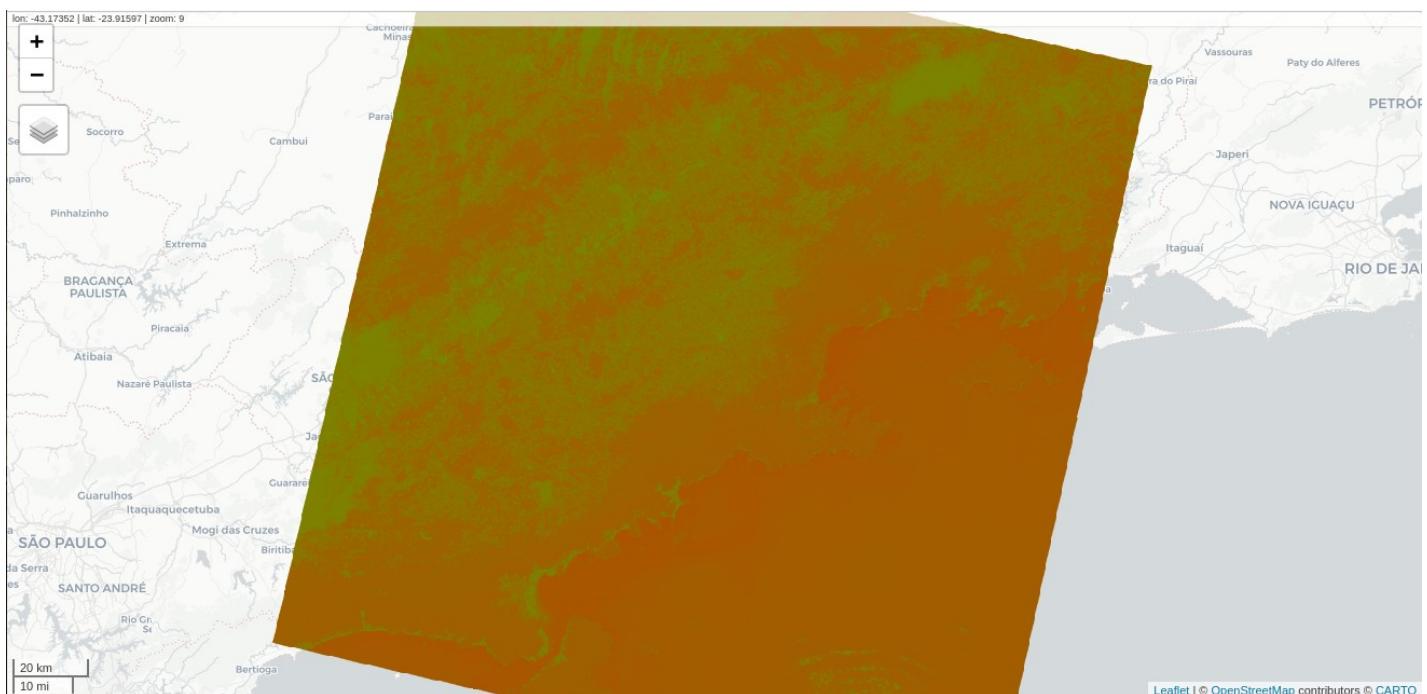
Map$centerObject(imagem, 9)
Map$addLayer(ndvi, list(min=-1, max=1, palette=c('#FF0000',
  '#00FF00')), 'NDVI')+
Map$addLayer(evi, list(min=-1, max=1, palette=c('#FF0000',
  '#00FF00')), 'EVI')

```

EVI



NDVI

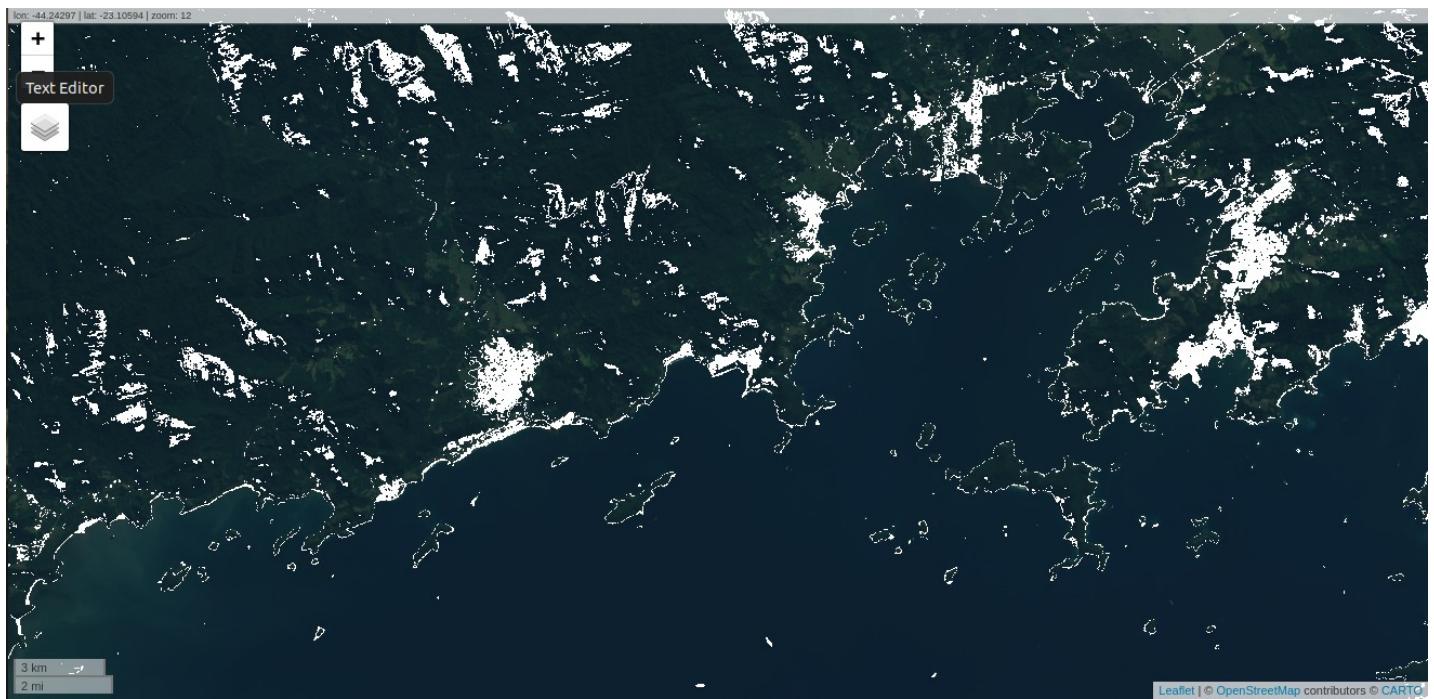


Operações relacionais, booleanas e condicionais com ee#Image

Operações destas categorias são muito usadas na criação de máscaras, filtragem e geração de 'heat maps'. Vamos aqui ver como podemos empregar essa operações.

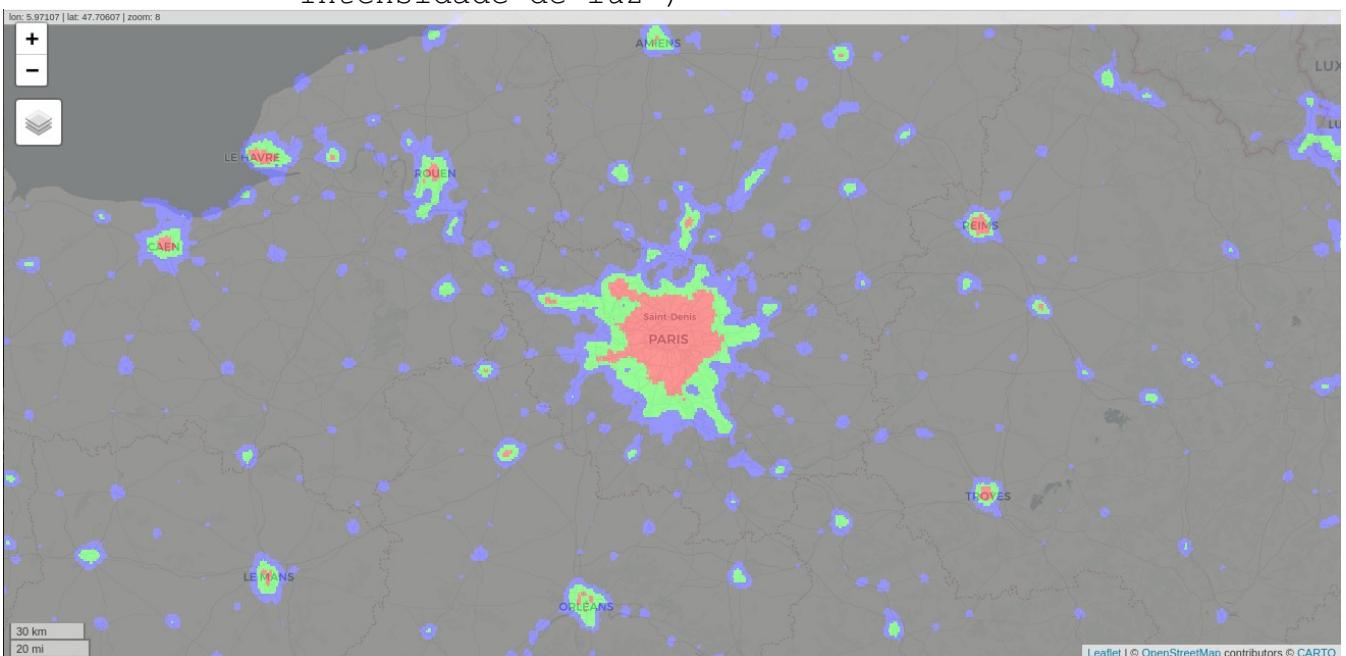
Criando uma máscara binária. Vamos criar duas imagens (NDWI e NDVI) e selecionar áreas que não são água e nem cobertura vegetal, ou seja, áreas expostas de rocha, solo, praia e zonas urbanas.

```
library(rgee)
ee_Initialize()
coleção<-ee$ImageCollection('LANDSAT/LC08/C01/T1_TOA')
ponto <- ee$Geometry$Point(-44.4678,-23.0058)
início <- ee>Date("2019-06-11")
fim <- ee>Date("2019-08-20")
filtro<-coleção$filterBounds(ponto)$filterDate(início,fim)
imagem <- filtro$first()
ndvi<-imagem$normalizedDifference(c('B5', 'B4'))
ndwi<- imagem$normalizedDifference(c('B3', 'B5'))
exposto<-ndvi$lt(0.5)$And(ndwi$lt(0.2))
Map$setCenter(-44.4678,-23.0058, 12)
imgViz<-list(min=0,max=0.5,bands=c("B4", "B3", "B2"), gamma=c(0.95, 1.1, 1))
Map$addLayer(imagem,imgViz, 'Landsat')+
  Map$addLayer(exposto$updateMask(exposto),list(),'Terreno exposto')
```



Paris é conhecida como a cidade luz, vamos checar usando com operações relacionais e condicionais.

```
library(rgee)
ee_Initialize()
nl2012<-ee$Image ('NOAA/DMSP-OLS/NIGHTTIME_LIGHTS/F182012')
paleta<-c('#000000', '#0000FF', '#00FF00', '#FF0000')
Map$setCenter(2.373, 48.8683, 8)
luz<-nl2012$expression(
  "(b('stable_lights') > 62) ? 3
   : (b('stable_lights') > 55) ? 2
   : (b('stable_lights') > 30) ? 1 : 0"
)
Map$addLayer(luz, list(
  min=0,max=3,palette=paleta,opacity=0.4),
  'Intensidade de luz')
```



Transformações espectrais com ee\$Image

Existem várias funções de transformação espectral no Earth Engine. Incluindo instâncias para imagens tais como `normalizedDifference()`, `unmix()`, `rgbToHsv()` and `hsvToRgb()`. As duas últimas podem ser usadas para efetuar um processo de pansharpening.

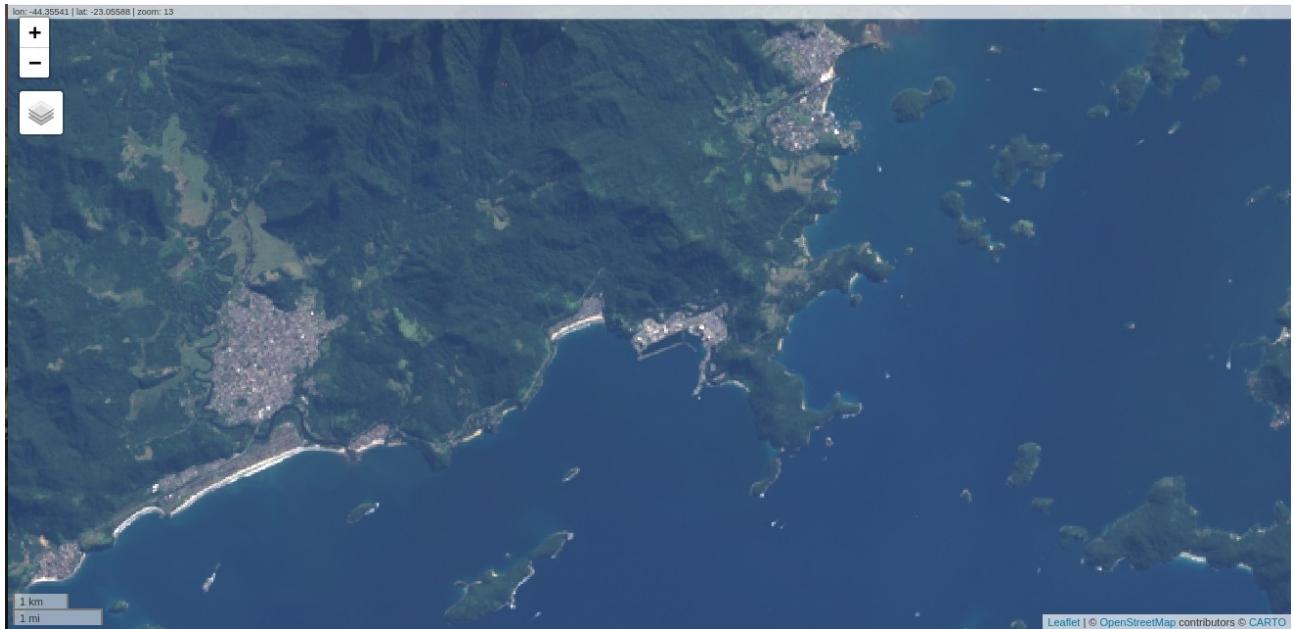
Pansharpening é um processo de juntar imagem pancromática de alta resolução com imagens multiespectrais de menor resolução para criar uma imagem colorida

Banda colorida de baixa resolução + Banda cinza de alta resolução
= Imagem colorida de alta resolução

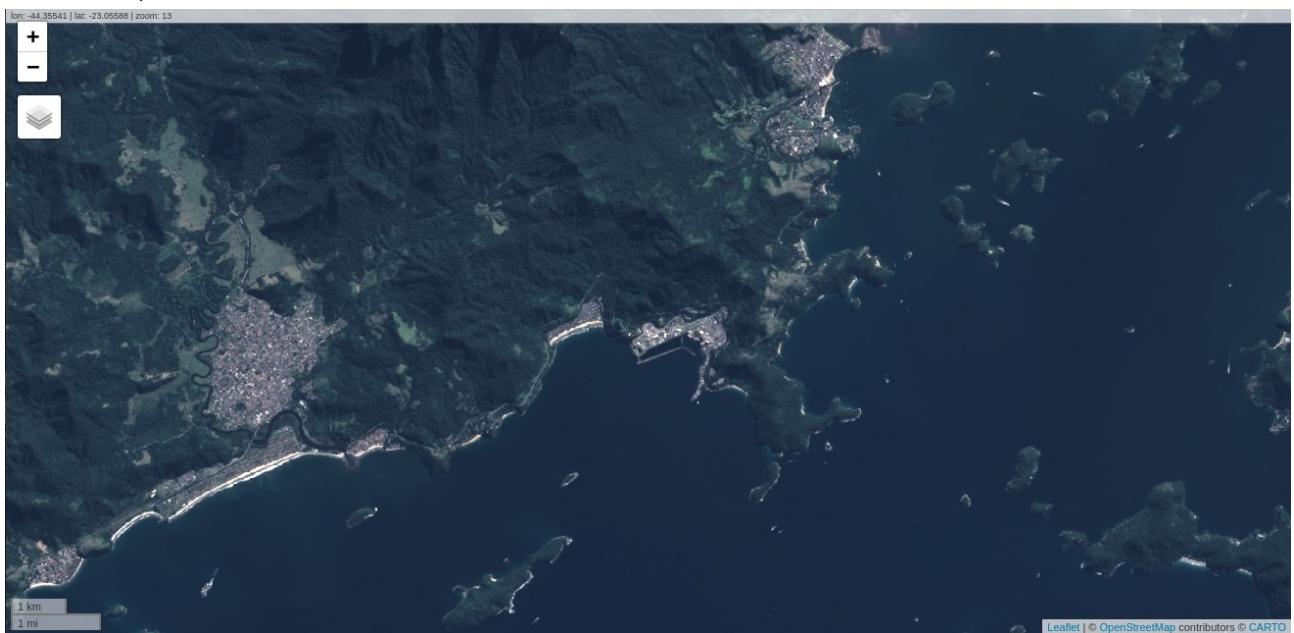
Isto é feito decompondo uma imagem RGB em HSV e substituindo a banda de valor (v) com a banda de alta resolução pancromática e convertendo de volta para RGB.

```
library(rgee)
ee_Initialize()
coleção<-ee$ImageCollection('LANDSAT/LC08/C01/T1_TOA')
ponto <- ee$Geometry$Point(-44.4678,-23.0058)
início <- ee>Date("2019-06-11")
fim <- ee>Date("2019-08-20")
filtro<-coleção$filterBounds(ponto)$filterDate(início,fim)
imagem <- filtro$first()
hsv<-imagem$select(c('B4', 'B3', 'B2'))$rgbToHsv()
sharpened<-ee$Image$cat(c(
  hsv$select('hue'), hsv$select('saturation'),
  imagem$select('B8'))
 )$hsvToRgb()
Map$setCenter(-44.4678,-23.0058, 13)
Map$addLayer(imagem, list(
  bands=c('B4', 'B3', 'B2'), min=0, max=0.25, gamma=c(1.3, 1.3,
  1.3)),
  'rgb')+
Map$addLayer(hsv, list(
  bands=c('hue', 'saturation', 'value'), min=0, max=1,
  gamma=c(1.3, 1.3, 1.3)),
  'hsv')+
Map$addLayer(sharpened, list(
  min=0, max=0.25, gamma=c(1.3, 1.3, 1.3)),
  'pansharpened')
```

O resultado será : RGB



Pensharpened



Previamente já vimos como usar a função `normalizedDifference()`.

Vamos agora ver como usar a função `unmix()` para separar e agrupar regiões de respostas espectrais semelhantes.

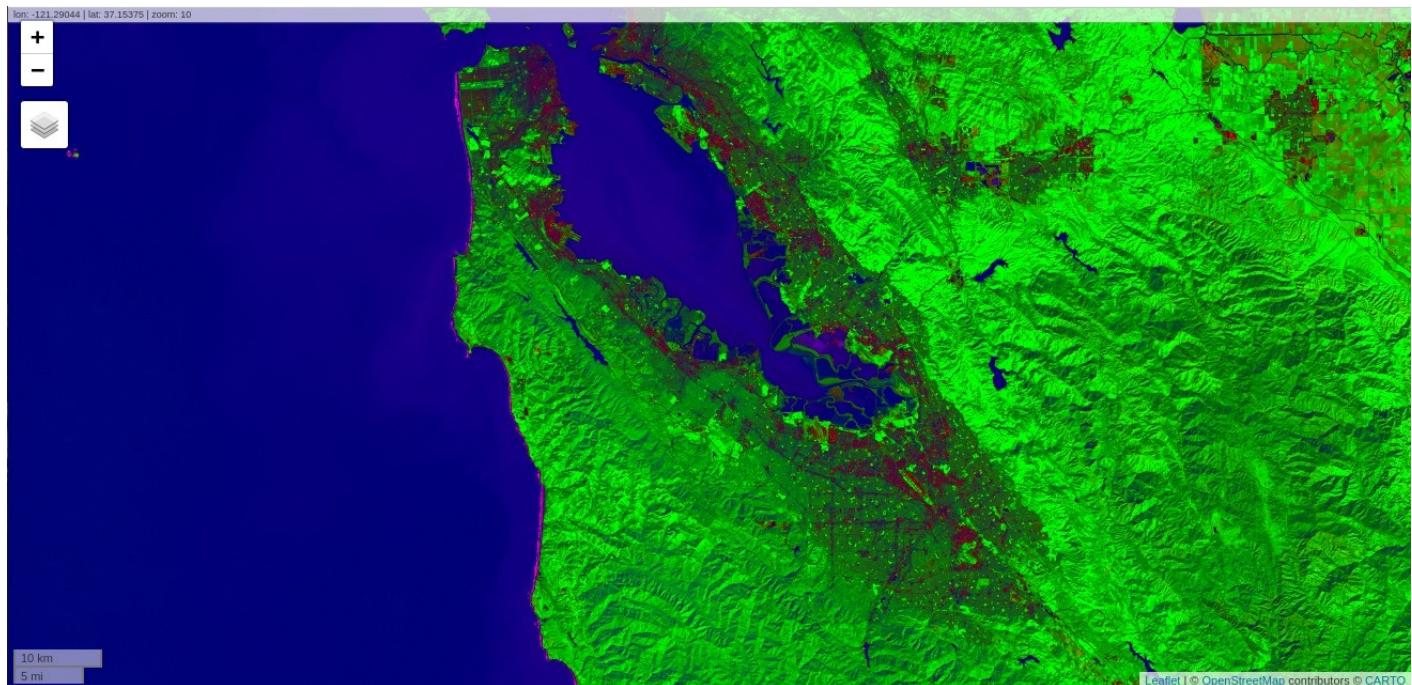
```
library(rgee)
ee_Initialize()
bandas <- c("B1", "B2", "B3", "B4", "B5", "B6", "B7")
imagem <- ee$Image("LANDSAT/LT05/C01/T1/LT05_044034_20080214")
$select(bandas)
# Definição dos valores finais espectrais de cada classe nas 7
# bandas.
```

```

urbano <- c(88, 42, 48, 38, 86, 115, 59)
vegetação <- c(50, 21, 20, 35, 50, 110, 23)
água <- c(51, 20, 14, 9, 7, 116, 4)
# Unmix a imagem.
fractions <- imagem$unmix(list(urbano, vegetação, água))
Map$setCenter(lon=-122.1899,lat=37.5010) # Bahia de San Francisco
Map$setZoom(zoom = 10)
Map$addLayer(
  eeObject = imagem,
  visParams = list(min = 0, max = 128, bands = c("B4", "B3",
"B2")),
  name = "Imagen"
) +
Map$addLayer(
  eeObject = fractions,
  visParams = list(min = 0, max = 2),
  name = "unmixed"
)

```

O resultado é (observe a separação de água, área urbana e área vegetada):

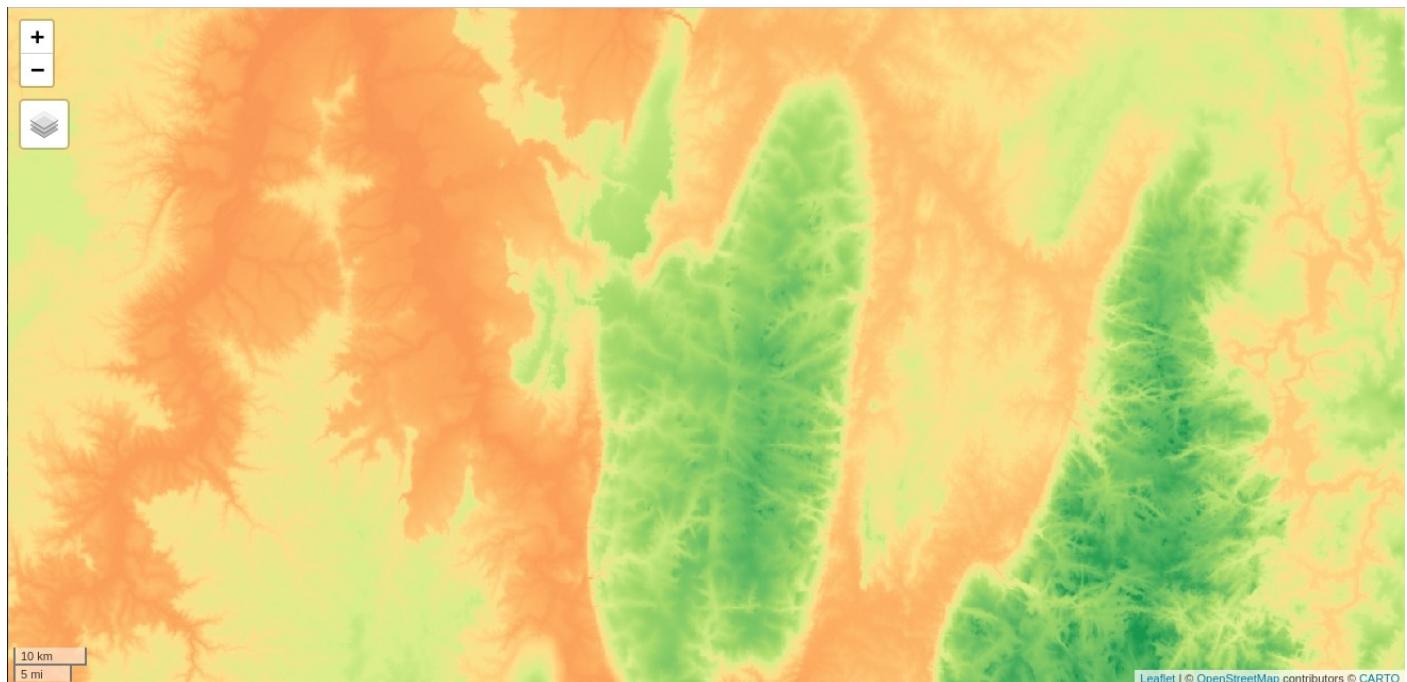


Gradiente de ee\$image

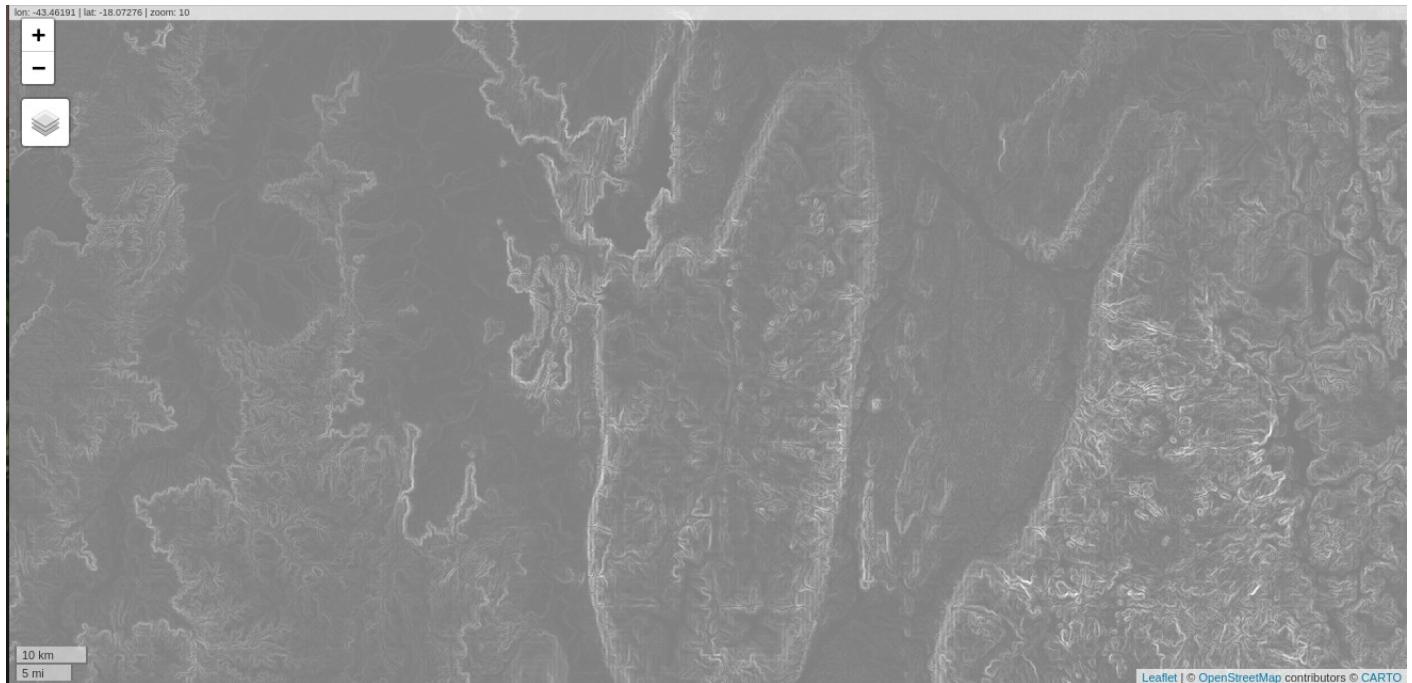
Vamos agora ver como extrair informações de gradient de imagem usando uma imagem DEM ALOS AW3D30 e vamos extrair o gradiente e a direção(aspecto).

```
library(rgee)
ee_Initialize()
dado <- ee$Image ("JAXA/ALOS/AW3D30_V1_1")
elevação <- dado$select ("AVE")
xyGrad <- elevação$gradient()
gradiente <- xyGrad$select ("x") $pow (2) $add (xyGrad$select ("y"))
$pow(2) $sqrt()
direção <- xyGrad$select ("y") $atan2 (xyGrad$select ("x"))
Map$setCenter (-44.366, -17.69, zoom = 10)
Map$addLayer(eeObject=direção, visParams=list(min=-2, max=2),
name="direção") +
Map$addLayer(eeObject=gradiente, visParams=list(min=-1, max=1),
name="gradiente") +
Map$addLayer(eeObject=elevação, visParams=list(palette = c(
"#d73027", "#f46d43", "#fdæe61", "#fee08b", "#d9ef8b", "#a6d96a",
"#66bd63", "#1a9850"), min=200, max=1400), name="Elevação")
```

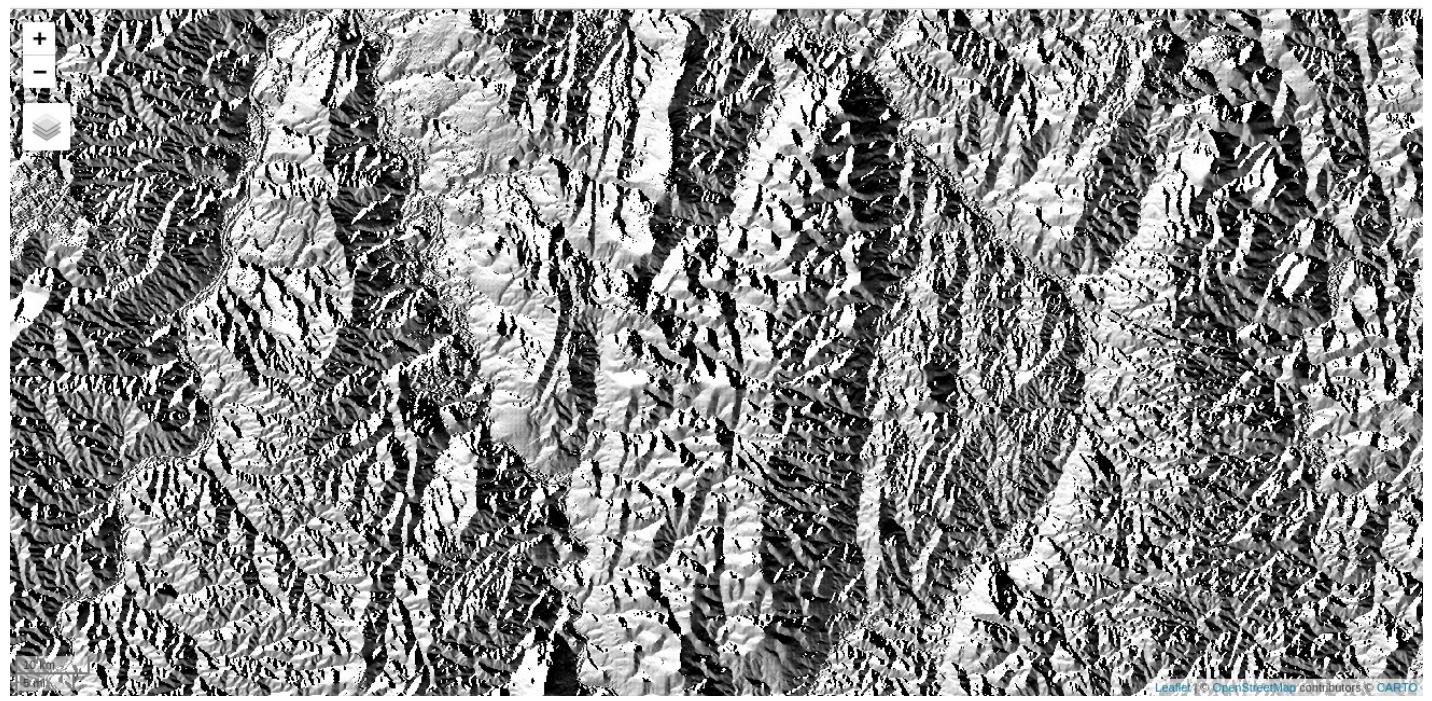
Vemos o resultado elevação



gradiente



direção ou aspecto

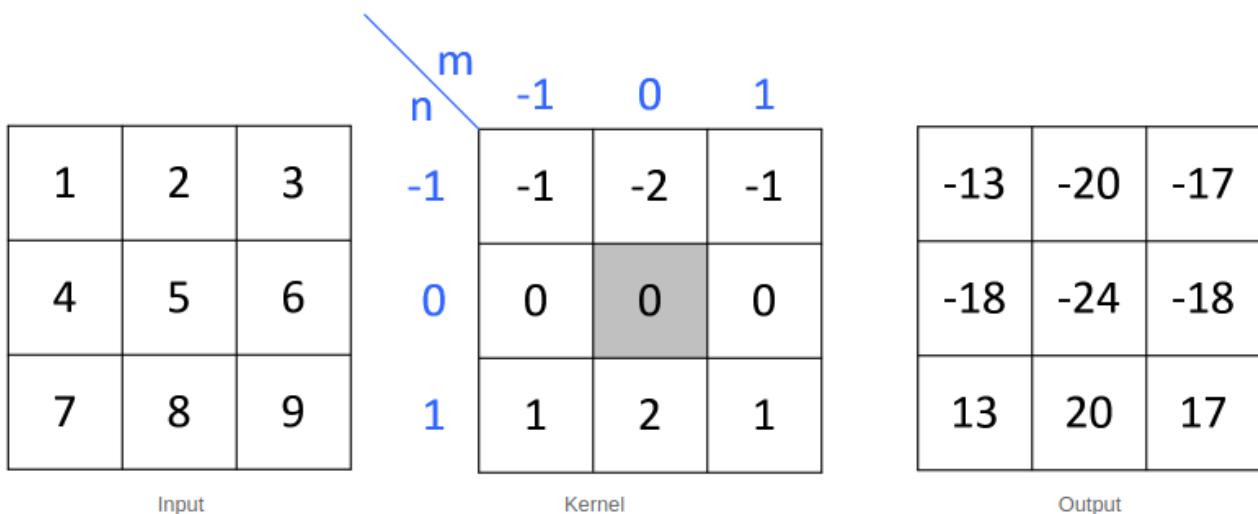


Convolução e detecção de bordas para ee\$Image

Na matemática convolução é uma operação envolvendo duas funções (f e g) que produz uma terceira função que expressa a forma que uma função é modificada pela outra. O termo se refere a ambos, a função resultante e processo de cálculo desta função. Em processamento digital de imagens a filtragem convolucional tem uma regra importante em algoritmos de detecção de bordas e processos relacionados. Ela é o resultado de um kernel (filtro modificador no formato de uma matriz $N \times M$) sobre os valores dos pixels da imagem.

1	2	1
0	0	0
-1	-2	-1
4	5	6

$$\begin{aligned}
 y[0,0] &= \sum_j \sum_i x[i,j] \cdot h[0-i, 0-j] \\
 &= x[-1, -1] \cdot h[1, 1] + x[0, -1] \cdot h[0, 1] + x[1, -1] \cdot h[-1, 1] \\
 &\quad + x[-1, 0] \cdot h[1, 0] + x[0, 0] \cdot h[0, 0] + x[1, 0] \cdot h[-1, 0] \\
 &\quad + x[-1, 1] \cdot h[1, -1] + x[0, 1] \cdot h[0, -1] + x[1, 1] \cdot h[-1, -1] \\
 &= 0 \cdot 1 + 0 \cdot 2 + 0 \cdot 1 \\
 &\quad + 0 \cdot 0 + 1 \cdot 0 + 2 \cdot 0 \\
 &\quad + 0 \cdot (-1) + 4 \cdot (-2) + 5 \cdot (-1) \\
 &= -13
 \end{aligned}$$



Diferentes métodos usam diferentes equações de interação da demonstrada acima. Vamos ver agora na prática como usar convolução e detectores de bordas.

```

library(rgee)
ee_Initialize()
coleção<-ee$ImageCollection('LANDSAT/LC08/C01/T1_TOA')
ponto <- ee$Geometry$Point(-44.4678,-23.0058)
início <- ee$Date("2019-06-11")
fim <- ee$Date("2019-08-20")
filtro<-coleção$filterBounds(ponto)$filterDate(início,fim)
imagem <- filtro$first()

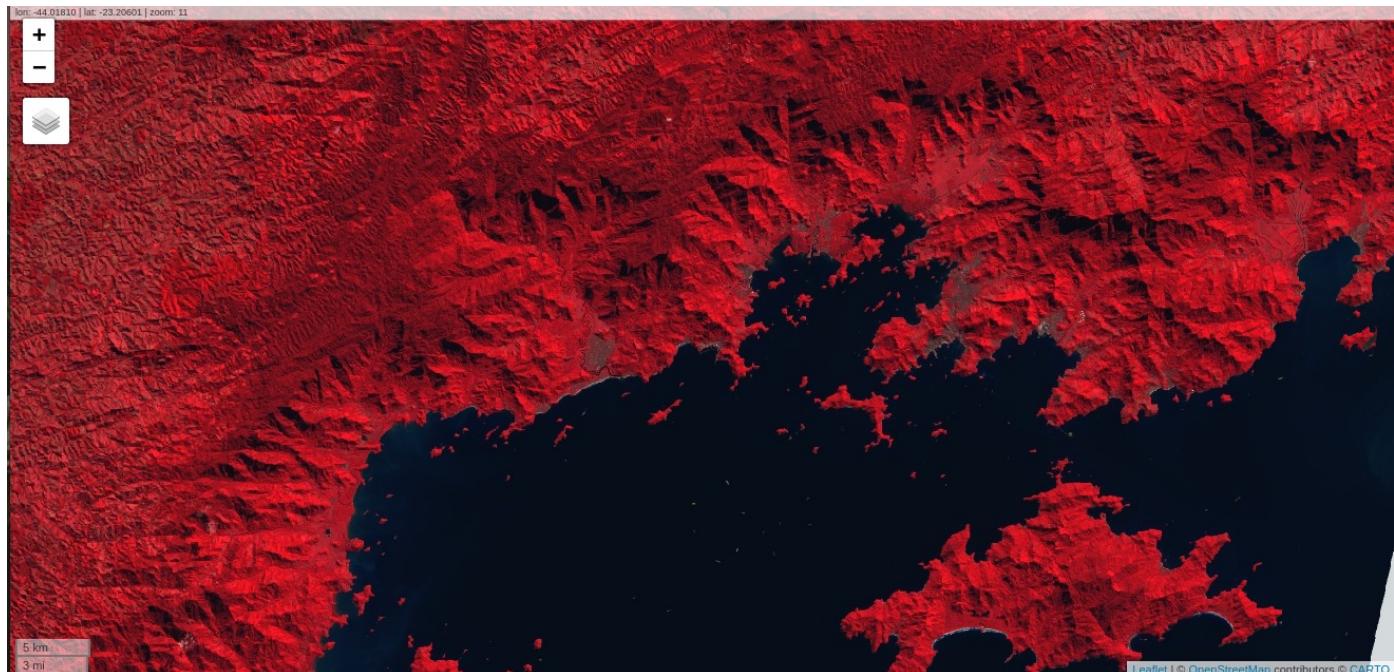
boxcar <- ee$Kernel$square(7, "pixels", T)
suavizado <- imagem$convolve(boxcar)

laplacian <- ee$Kernel$laplacian8(1, F)
acentuado <- imagem$convolve(laplacian)

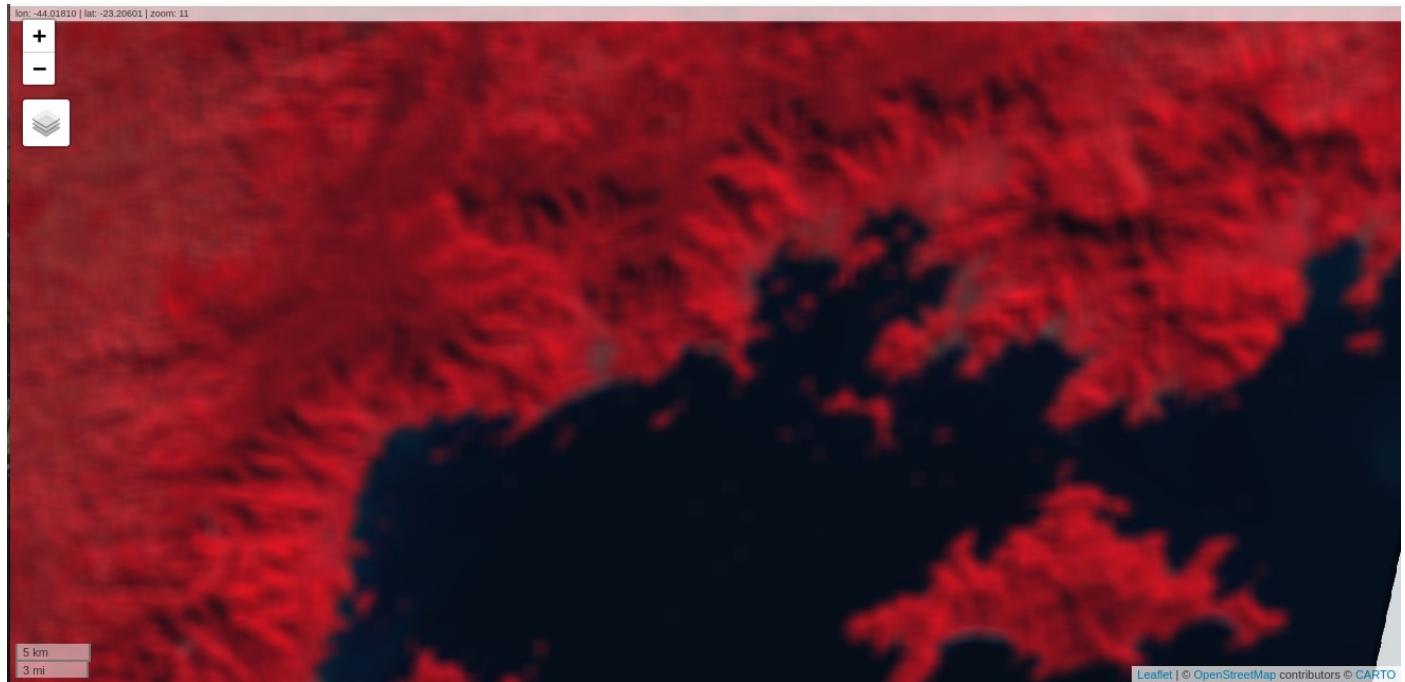
Map$setCenter(-44.4678,-23.0058)
Map$setZoom(zoom = 11)
Map$addLayer(
  eeObject = imagem,
  visParams = list(bands = c("B5", "B4", "B3"), max = 0.5),
  name = "Imagen"
) +
Map$addLayer(
  eeObject = suavizado,
  visParams = list(bands = c("B5", "B4", "B3"), max = 0.5),
  name = "Suavizado"
) +
Map$addLayer(
  eeObject = acentuado,
  visParams = list(bands = c("B5", "B4", "B3"), max = 0.5),
  name = "Acentuado"
)

```

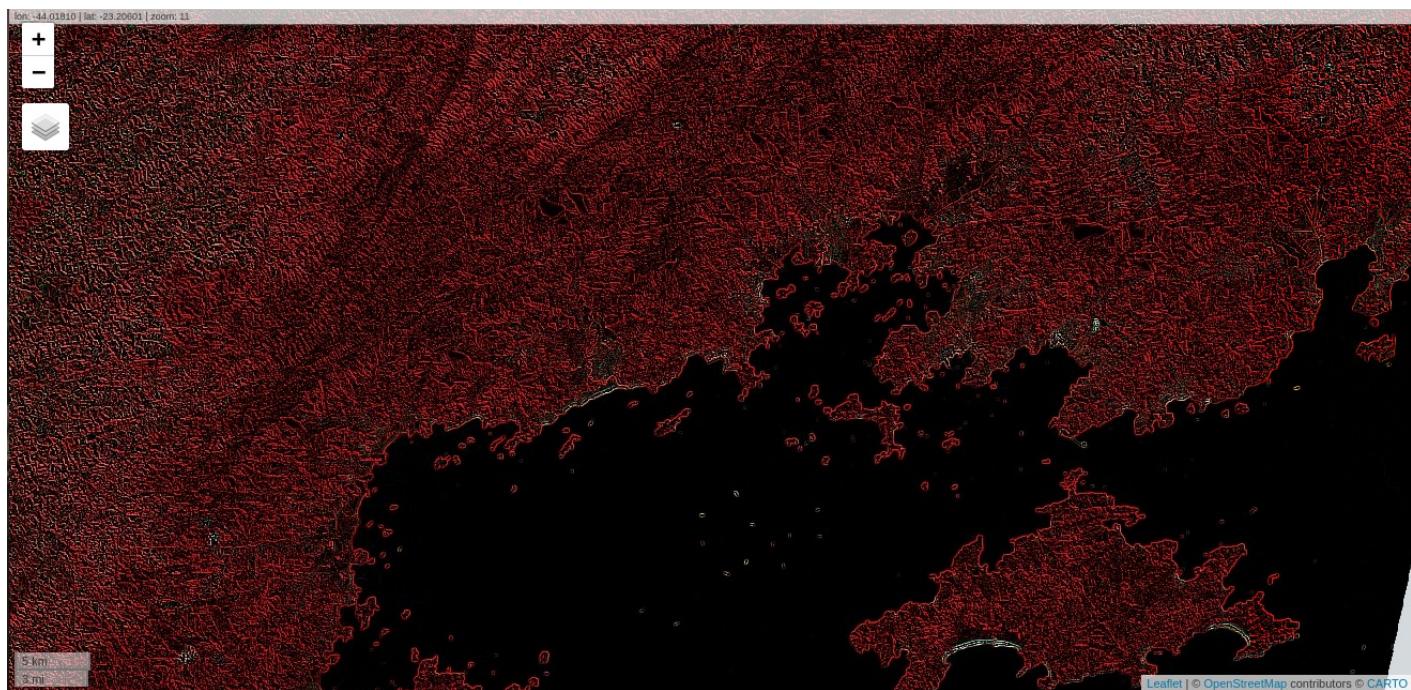
Imagen Original (Bandas 5, 4 e 3)



Suavizado



Acentuado



Podemos criar o nosso próprio kernel com pesos e formas arbitrariamente definidos, para o tal use `ee.Kernel.fixed()`. Por exemplo, o código abaixo cria um kernel 9x9 de valores 1 com um zero no meio;


```

        1.0,
        1.0,
        1.0,
        1.0,
        1.0,
        1.0
    ],
    [
        1.0,
        1.0,
        1.0,
        1.0,
        1.0,
        1.0,
        1.0,
        1.0,
        1.0
    ],
    [
        1.0,
        1.0,
        1.0,
        1.0,
        1.0,
        1.0,
        1.0,
        1.0,
        1.0
    ],
    [
        1.0,
        1.0,
        1.0,
        1.0,
        1.0,
        1.0,
        1.0,
        1.0,
        1.0
    ],
    [
        1.0,
        1.0,
        1.0,
        1.0,
        1.0,
        1.0,
        1.0,
        1.0,
        1.0
    ],
    [
        1.0,
        "x": -4.0,
        "y": -4.0,
        "normalize": false
    },
    "functionName": "Kernel.fixed"
)

```

Vamos agora ver alguns algoritmos de detecção de borda.

```

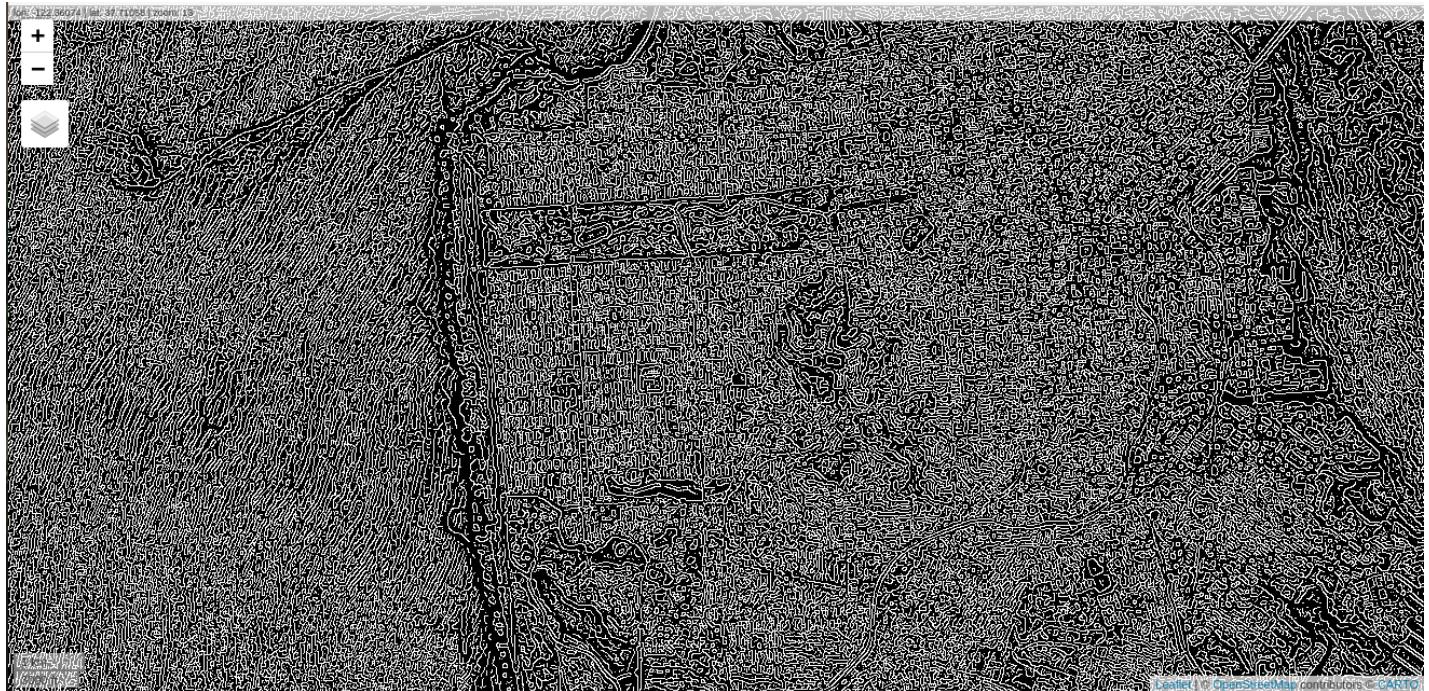
library(rgee)
ee_Initialize()
imagem <- ee$Image ("LANDSAT/LC08/C01/T1/LC08_044034_20140318")
$select("B8")
canny <- ee$Algorithms$CannyEdgeDetector(image=imagem, threshold =
1, sigma = 1)
hough <- ee$Algorithms$HoughTransform(canny, 256, 600, 100)
fat <- ee$Kernel$gaussian(
    radius = 3,
    sigma = 3,
    units = "pixels",
    normalize = T,
    magnitude = -1
)
skinny <- ee$Kernel$gaussian(
    radius = 3,
    sigma = 1,
    units = "pixels",

```

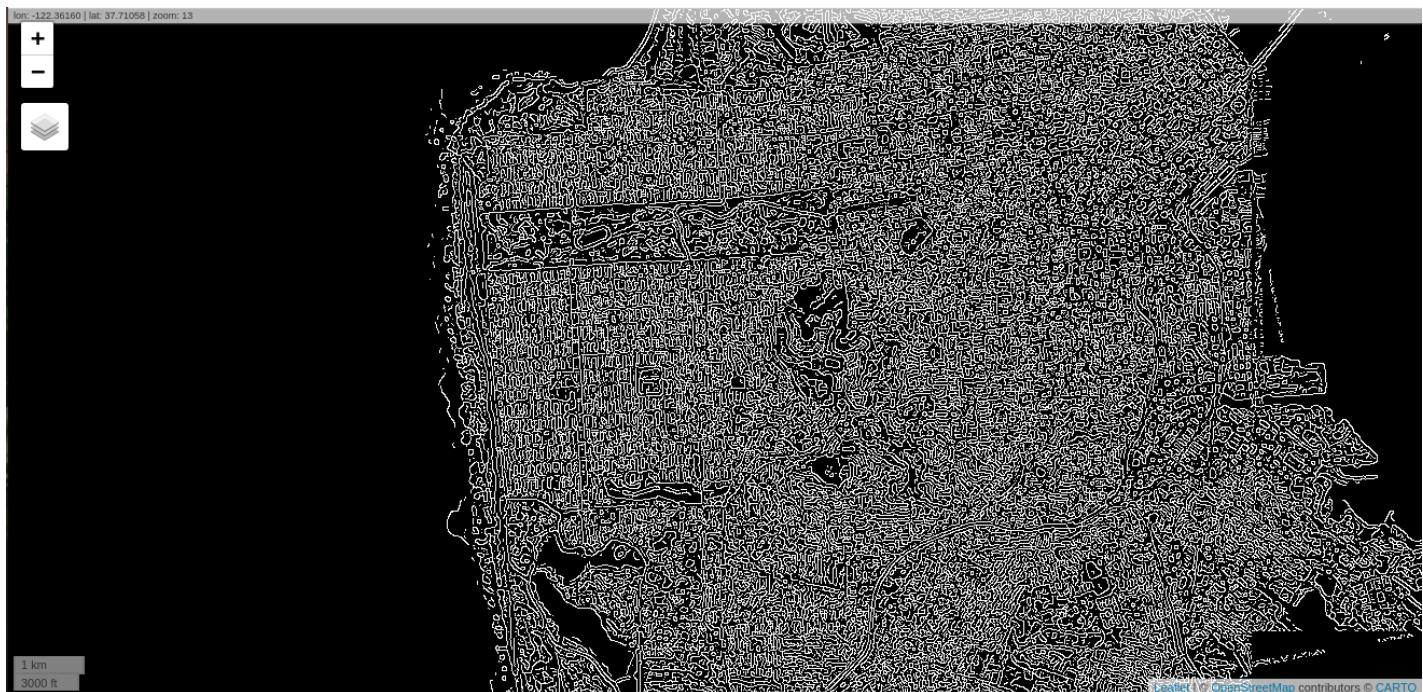
```
normalize = T
)
dog <- fat$add(skinny)
zeroXings <- imagem$convolve(dog)$zeroCrossing()

Map$setCenter(-122.054, 37.7295)
Map$setZoom(zoom = 10)
Map$addLayer(
  eeObject = canny,
  visParams = list(),
  name = "canny"
) +
  Map$addLayer(
    eeObject = hough,
    visParams = list(),
    name = "hough"
) +
  Map$addLayer(
    eeObject = imagem,
    visParams = list(gamma=1.5),
    name = "Landsat B8"
) +
  Map$addLayer(
    eeObject = zeroXings$updateMask(zeroXings),
    visParams = list(palette = "FF0000"),
    name = "zero crossings"
)
```

Canny



Hough



Zero Crossing



ee\$ImageCollection

Construtores de ImageCollection

Um objeto ee\$Image Collection é uma pilha ou sequência de imagens. Este objeto pode ser carregando colando uma ID de um ativo no construtor do objeto ee\$ImageCollection.

```
library(rgee)
ee_Initialize()
coleção<-ee$ImageCollection('COPERNICUS/S2_SR')
```

Essa coleção contém uma referência para todas imagens Sentinel 2 no catálogo público e são muitas imagens, geralmente usamos filtros associados a coleções de imagens.

```
ponto <- ee$Geometry$Point(-44.366,-18.145)
início <- ee>Date("2019-07-11")
fim <- ee>Date("2019-07-20")
coleção.filt<-coleção$filterBounds(ponto)$filterDate(início,fim)
```

O Earth Engine também possui funções para criar um object Image Collection. O construtor ee \$ImageCollection() e os métodos ee\$ImageCollection\$fromImages podem resultar em um novo objeto.

```
imagem1<-ee$Image(1)
imagem2<-ee$Image(2)
coleção1<-ee$ImageCollection(list(imagem1,imagem2))
ee_print(coleção1)
----- Earth Engine ImageCollection -----
ImageCollection Metadata:
- Class : ee$ImageCollection
- Number of Images : 2
- Number of Properties : 0
- Number of Pixels* : 129600
- Approximate size* : 202.50 KB
Image Metadata (img_index = 0):
- ID : no_id
- Number of Bands : 1
- Bands names : constant
- Number of Properties : 1
- Number of Pixels* : 64800
- Approximate size* : 101.25 KB
Band Metadata (img_band = 'constant'):
- EPSG (SRID) : 4326
- proj4string : +proj=longlat +datum=WGS84 +no_defs
- Geotransform : 1 0 0 0 1 0
- Nominal scale (meters) : 111319.5
- Dimensions : 360 180
- Number of Pixels : 64800
- Data type : INT
- Approximate size : 101.25 KB
```

NOTE: (*) Properties calculated considering a constant geotransform and data type.

Outras formas de criarmos coleções:

```
coleção2<-ee$ImageCollection$fromImages(list(ee$Image(3),
                                                ee$Image(4)))
ee_print(coleção2)
_____
Earth Engine ImageCollection —
```

ImageCollection Metadata:

- Class : ee\$ImageCollection
- Number of Images : 2
- Number of Properties : 0
- Number of Pixels* : 129600
- Approximate size* : 202.50 KB

Image Metadata (img_index = 0):

- ID : no_id
- Number of Bands : 1
- Bands names : constant
- Number of Properties : 1
- Number of Pixels* : 64800
- Approximate size* : 101.25 KB

Band Metadata (img_band = 'constant'):

- EPSG (SRID) : 4326
- proj4string : +proj=longlat +datum=WGS84 +no_defs
- Geotransform : 1 0 0 0 1 0
- Nominal scale (meters) : 111319.5
- Dimensions : 360 180
- Number of Pixels : 64800
- Data type : INT
- Approximate size : 101.25 KB

NOTE: (*) Properties calculated considering a constant geotransform and data type.

Podemos também unir duas coleções de imagem usando o método \$merge.

```
coleção3<-coleção2$merge(coleção1)
ee_print(coleção3)
_____
Earth Engine ImageCollection —
```

ImageCollection Metadata:

- Class : ee\$ImageCollection
- Number of Images : 4
- Number of Properties : 0
- Number of Pixels* : 259200
- Approximate size* : 405.00 KB

Image Metadata (img_index = 0):

- ID : no_id
- Number of Bands : 1
- Bands names : constant
- Number of Properties : 1
- Number of Pixels* : 64800
- Approximate size* : 101.25 KB

Band Metadata (img_band = 'constant'):

- EPSG (SRID) : 4326
- proj4string : +proj=longlat +datum=WGS84 +no_defs
- Geotransform : 1 0 0 0 1 0
- Nominal scale (meters) : 111319.5
- Dimensions : 360 180
- Number of Pixels : 64800
- Data type : INT
- Approximate size : 101.25 KB

NOTE: (*) Properties calculated considering a constant geotransform and data type.

E também podemos criar uma coleção usando imagens já existentes a partir de geotiffs numa cloud storage.

```
uri1<-'gs://gcp-public-data-landsat/LT05/01/001/001/
LT05_L1GS_001001_19910509_20180220_01_T2/
LT05_L1GS_001001_19910509_20180220_01_T2_B5.TIF'
uri2<-'gs://gcp-public-data-landsat/LT05/01/001/001/
LT05_L1GS_001001_19910509_20180220_01_T2/
LT05_L1GS_001001_19910509_20180220_01_T2_B4.TIF'
uri3<-'gs://gcp-public-data-landsat/LT05/01/001/001/
LT05_L1GS_001001_19910509_20180220_01_T2/
LT05_L1GS_001001_19910509_20180220_01_T2_B3.TIF'
uri4<-'gs://gcp-public-data-landsat/LT05/01/001/001/
LT05_L1GS_001001_19910509_20180220_01_T2/
LT05_L1GS_001001_19910509_20180220_01_T2_B2.TIF'
imagens1<-ee$Image$loadGeoTIFF(uri1)
imagens2<-ee$Image$loadGeoTIFF(uri2)
imagens3<-ee$Image$loadGeoTIFF(uri3)
imagens4<-ee$Image$loadGeoTIFF(uri4)
coleção<-
ee$ImageCollection$fromImages(list(imagens1, imagens2, imagens3, imagens4))
rgb<-coleção$toBands()$rename(c('B5', 'B4', 'B3', 'B2'))
Map$centerObject(rgb)
Map$addLayer(rgb, list(bands=c('B5', 'B3', 'B2'), min=0, max= 1000), 'rgb')
```

Filtrando ImageCollection

A filtragem de uma coleção de imagens é um ponto bastante importante quando trabalhamos com `ImageCollections`. Com ela restringimos os resultados para um área ou tipo que especificamos através de filtros. Vamos ver um exemplo onde filtramos imagens por data e área bem como por valor de metadados para qualidade.

```
coleção<-ee$ImageCollection('LANDSAT/LT05/C01/T2')$filterDate('1987-01-
01','1990-05-01')$filterBounds(ee$Geometry$Point(25.8544,-18.08874))
filtrado <- coleção$filterMetadata('IMAGE_QUALITY', 'equals', 9)
compRuim<-ee$Algorithms$Landsat$simpleComposite(coleção, 75, 3)
compBoa<-ee$Algorithms$Landsat$simpleComposite(filtrado, 75, 3)
Map$setCenter(25.8544,-18.08874,13)
Map$addLayer(compRuim,
             list(bands=c('B3', 'B2', 'B1'), gain=3.5),
             'Composição Ruim')
Map$addLayer(compBoa,
             list(bands=c('B3', 'B2', 'B1'), gain=3.5),
             'Composição Boa')
```

Informações e metadata de ImageCollection

Podemos usar funções para verificar informações gerais e de metadata de uma coleção de imagens.

```
coleção<-ee$ImageCollection('LANDSAT/LC08/C01/T1_TOA')
$filter(ee$Filter$eq('WRS_PATH', 44))$filter(ee$Filter$eq('WRS_ROW', 34))
$filterDate('2014-03-01', '2014-08-01')
ee_print(coleção) — Earth Engine ImageCollection —
ImageCollection Metadata:
- Class : ee$ImageCollection
- Number of Images : 9
```

```

- Number of Properties : 41
- Number of Pixels*   : 6435271800
- Approximate size*   : 111.20 GB
Image Metadata (img_index = 0):
- ID                  : LANDSAT/LC08/C01/T1_TOA/LC08_044034_20140318
- Time start          : 2014-03-18 18:46:32
- Number of Bands     : 12
- Bands names         : B1 B2 B3 B4 B5 B6 B7 B8 B9 B10 B11 BQA
- Number of Properties: 118
- Number of Pixels*   : 715030200
- Approximate size*   : 12.36 GB
Band Metadata (img_band = 'B1'):
- EPSG (SRID)         : 32610
- proj4string : +proj=utm +zone=10 +datum=WGS84 +units=m +no_defs
- Geotransform        : 30 0 460785 0 -30 4264215
- Nominal scale (meters): 30
- Dimensions          : 7650 7789
- Number of Pixels    : 59585850
- Data type           : FLOAT
- Approximate size    : 1.03 GB

```

Com a função `ee_print()` obtemos um grande número de informações de metadata da coleção. Também podemos acessar informações específicas com as funções:

```

# Obtém o número de conjunto de imagens na coleção.
tamanho<-coleção$size()
cat("Tamanho: ", tamanho$getInfo(), '\n')

#Obtém a extensão de datas das imagens na coleção
range<-coleção$reduceColumns(ee$Reducer$minMax(),list("system:time_start"))
col_min <- eedate_to_rdate(range$get("min"))
col_max <- eedate_to_rdate(range$get("max"))
cat("Date range: ", as.character(col_min),as.character(col_max),'\n')

# Obtém estatísticas de uma propriedade (elevação solar) nesta coleção.
sunStats <- coleção$aggregate_stats("SUN_ELEVATION")
sunStats$getInfo()

# Ordena a coleção pela cobertura de nuvens, obtém a com menor cobertura.
image <- ee$Image(coleção$sort("CLOUD_COVER")$first())
image$getInfo()

# Limita atcoleção para as 10 imagens mais recentes.
recent <- coleção$sort("system:time_start", FALSE)$limit(10)
recent$getInfo()

```

Visualizações com ImageCollection

Vamos agora ver como podemos interagir com coleções de imagens para gerar visualizações na forma de animações.

Nesse primeiro exemplo usamos o MODIS com composição NDVI. Selecionamos a região do Nordeste do Brasil e criamos uma animação entre primeiro de janeiro de 2013 e primeiro de janeiro de 2014.

```

coleção<-ee$ImageCollection('MODIS/006/MOD13A2')$select('NDVI')
máscara<-ee$FeatureCollection('USDOS/LSIB_SIMPLE/2017')
$filter(ee$Filter$eq('wld_rgn', 'South America'))
região<-ee$Geometry$Polygon(list(c(-48.0, 0.0),
                                    c(-48.0, -17.0),c(-34.0, -17.0),c(-34.0, 0.0)))
coleção<-coleção$map(function(img) {

```

```

doy<-ee$Date(img$get('system:time_start'))$getRelative('day', 'year')
return (img$set('doy', doy)))
distinctDOY<-coleção$filterDate('2013-01-01', '2014-01-01')
filter<-ee$filter$equals(leftField='doy', rightField= 'doy')
join<-ee$Join$saveAll('doy_matches')
joinCol<-ee$imageCollection(join$apply(distinctDOY, coleção, filter))
comp<-joinCol$map(function(img) {
  doyCol<-ee$imageCollection$fromImages(img$get('doy_matches'))
  return (doyCol$reduce(ee$Reducer$median())))
imagens<-comp$map(function(img) {
  return (img$visualize('NDVI_median')$clip(máscara)))})
gifParams<-list('region'=região, 'dimensions'=600, 'crs'='EPSG:3857',
  'framesPerSecond'=5, palette=c(
    '#FFFFFF', '#CE7E45', '#DF923D', '#F1B555', '#FCD163', '#99B718', '#74A901',
    '#66A000', '#529400', '#3E8601', '#207401', '#056201', '#004C00', '#023B01',
    '#012E01', '#011D01', '#011301'), min=110.0, max=250.0)
print(imagens$getVideoThumbURL(gifParams))

```

O resultado será um link para uma animação gif.

Clique no link para ver o resultado <http://amazeone.com.br/barebra/pandora/ne.gif>

Vamos agora criar uma animação com a variação de temperatura entre 20 e 23 de junho de 2018 na America do Sul.

```

aoi<-ee$Geometry$Polygon(list(c(-90,15),c(-90,-55),c(-35,-55),c(-35,15)))
tempCol<-ee$imageCollection('NOAA/GFS0P25')$filterDate('2018-06-20', '2018-06-23')$limit(72)$select('temperature_2m_above_ground')
videoArgs<-list(dimensions=600, region=aoi, framesPerSecond=7, crs='EPSG:3857',
  min=-40.0, max=35.0, palette=c('blue','purple','cyan','green','yellow','red'))
print(tempCol$getVideoThumbURL(videoArgs))

```

O resultado será um link para uma animação gif.

Clique no link para ver o resultado <http://amazeone.com.br/barebra/pandora/sa.gif>

Visualizando cada cena individualmente como uma fita de filme.

```

filmArgs<-list(dimensions=256, region=aoi, crs='EPSG:3857',
  min=-40.0, max=35.0, palette=c('blue','purple','cyan','green','yellow','red'))
print(tempCol$getFilmstripThumbURL(filmArgs))

```

O resultado será um link para a sequência de imagens.

Clique para ver o resultado <http://amazeone.com.br/barebra/pandora/filmstrip.png>

Técnicas avançadas de Visualização de ImageCollection

Estendendo um pouco o que pode ser feito com coleções para criar efeitos diferentes na visualização final.

Adicionando um objeto feature (veremos mais sobre ee\$Feature adiante).

```
tempCol<-ee$ImageCollection('NOAA/GFS0P25')$filterDate('2018-06-22','2018-06-23')$limit(24)$select('temperature_2m_above_ground')
tempColVis<-tempCol$map(function(img){return (img)})
southAmCol<-ee$FeatureCollection('USDOS/LSIB_SIMPLE/2017')
$filterMetadata('wld_rgn','equals','South America')
southAmAoi<-ee$Geometry$Rectangle(-90.6,-54.8,-28.4,17.4)
```

Agora vamos criar overlays associados a ImageCollection.

```
empty<-ee$Image()$byte()
southAmOutline<-empty$paint(featureCollection=southAmCol,color=1,width=1)$visualize(palette='#000000')
tempColOutline<-tempColVis$map(function(img) {return (img$blend(southAmOutline))})
videoArgs<-list(dimensions=600,region=southAmAoi,framesPerSecond=7,
crs='EPSG:3857',bands=c('vis-red'),min=-40.0, max=35.0,palette=c('blue',
'purple', 'cyan', 'green', 'yellow', 'red'))
print(tempColOutline$getVideoThumbURL(videoArgs))
```

O resultado será um link para uma animação gif.

Link para ver o resultado http://amazeone.com.br/barebra/pandora/sa_adv1.gif

Vamos agora criar uma máscara sobre nossa animação

```
empty<-ee$Image()$byte()
southAmOutline<-empty$paint(featureCollection=southAmCol,color=1,width=1)$visualize(palette='#000000')
tempColOutline<-tempColVis$map(function(img) {return (img$blend(southAmOutline))})
dullLayer<-
ee$image$constant(200)$visualize(opacity=0.55,min=0,max=255,forceRgbOutput=TRUE)
finalVisCol<-tempColOutline$map(function(img){return (img$blend(dullLayer)
$blend(img$clipToCollection(southAmCol)))})
videoArgs<-
list(dimensions=600,region=southAmAoi,framesPerSecond=7,crs='EPSG:3857',bands=c(
'vis-red'),min=-40.0, max=35.0,palette=c('blue', 'purple', 'cyan', 'green',
'yellow', 'red'))
print(finalVisCol$getVideoThumbURL(videoArgs))
```

O resultado será um link para uma animação gif.

Link para ver o resultado http://amazeone.com.br/barebra/pandora/sa_adv2.gif

Vamos agora adicionar uma sombra de relevo (Hillshade) na animação.

```
hillshade<-ee$Terrain$hillshade(ee$image('USGS/SRTMGL1_003')$multiply(100))
$clipToCollection(southAmCol)
finalVisCol<-tempColVis$map(function(img) {
  return (hillshade$blend(img$clipToCollection(southAmCol)
$visualize(opacity=0.5,palette=c('blue', 'purple', 'cyan', 'green', 'yellow',
'red'),min=-40,max=35)))})
videoArgs <-list(dimensions=600,region=southAmAoi,framesPerSecond=7,
crs='EPSG:3857')
print(finalVisCol$getVideoThumbURL(videoArgs))
```

O resultado será um link para uma animação gif.

Link para ver o resultado http://amazeone.com.br/barebra/pandora/sa_adv3.gif

Por último veremos como fazer transição de imagens usando coleções.

Mostraremos os métodos 'flicker'

Primeiro criamos a coleção e usaremos uma máscara com as fronteiras de países.

```
aoi<-ee$Geometry$Polygon(list(c(-90,15),c(-90,-55),c(-35,-55),c(-35,15)))
temp<-ee$imageCollection('NOAA/GFSOP25')
#verão no hemisfério norte
summerSolTemp<-temp$filterDate('2018-06-21', '2018-06-22')
$filterMetadata('forecast_hours', 'equals', 12)$first()
$select('temperature_2m_above_ground')
#inverno no hemisfério norte
winterSolTemp<-temp$filterDate('2018-12-22', '2018-12-23')
$filterMetadata('forecast_hours', 'equals', 12)$first()
$select('temperature_2m_above_ground')
#combinando as duas coleções
tempCol<-ee$imageCollection(list(summerSolTemp$set('season',
'summer'),winterSolTemp$set('season', 'winter')))
countries<-ee$FeatureCollection('USDOS/LSIB_SIMPLE/2017')
tempColVis<-tempCol$map(function(img) {
  return (img$clipToCollection(countries)$unmask(0)$copyProperties(img,
img$propertyNames()))})
```

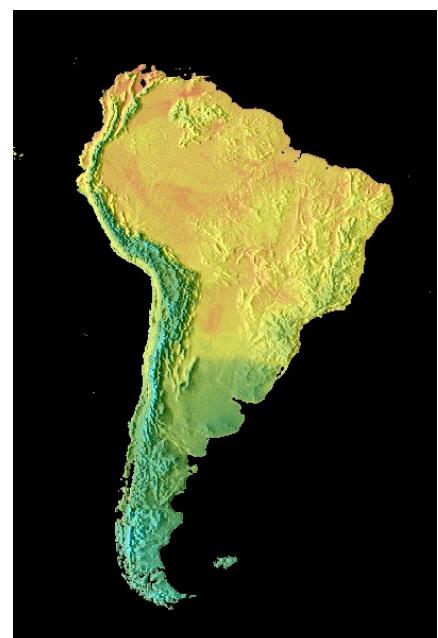
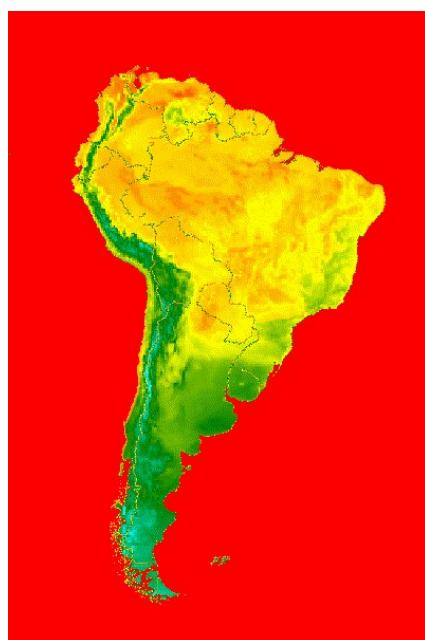
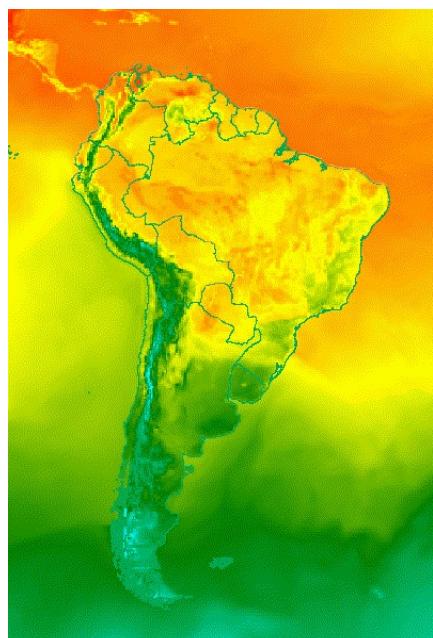
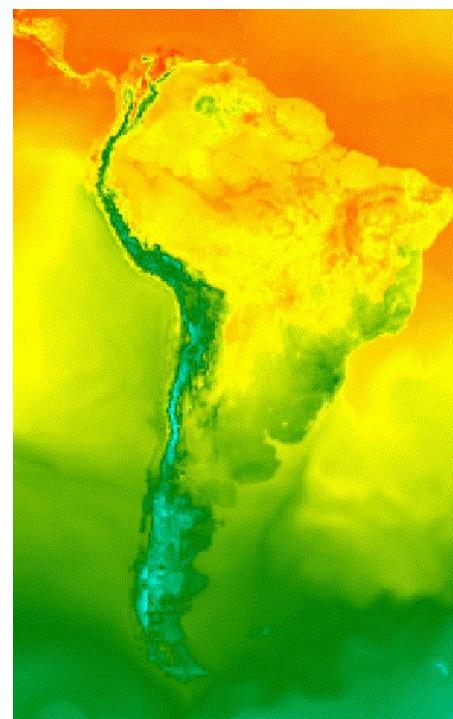
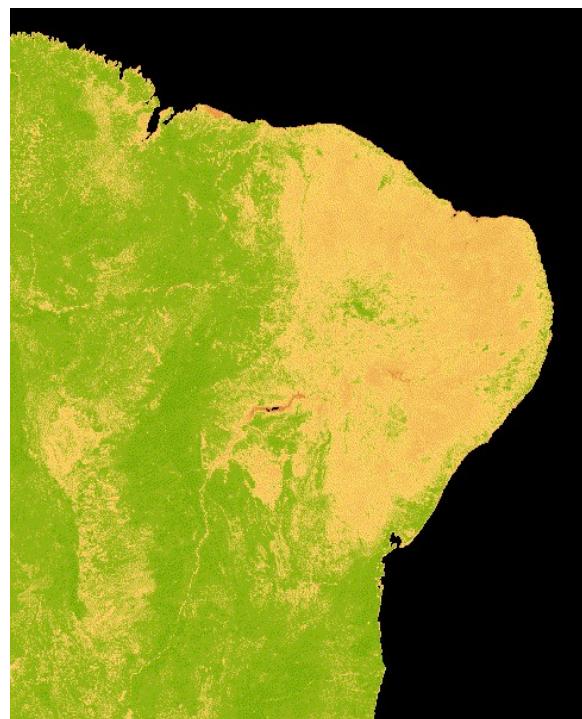
Para criar uma visualização flicker usamos:

```
videoArgs<-list(dimensions=600,region=aoi,framesPerSecond=2,crs='EPSG:3857',
bands=c('temperature_2m_above_ground'),min=-40.0, max=35.0,palette=c('blue',
'purple', 'cyan', 'green', 'yellow', 'red'))
print(tempColVis$getVideoThumbURL(videoArgs))
```

O resultado será um link para uma animação gif.

Link para ver o resultado http://amazeone.com.br/barebra/pandora/sa_trans1.gif

Alguns Snapshots dos Resultados da animações



ee\$Geometry

O earth engine lida com dados vetoriais na forma de um objeto ee\$Geometry. Eles são criados no formato geoJSON. Para maiores informações sobre o formato geoJSON visite <https://tools.ietf.org/html/rfc7946>.

```
point<-ee$Geometry$Point(c(1.5, 1.5))
point
ee.Geometry({
  "type": "Point",
  "coordinates": [
    1.5,
    1.5
  ]
})
lineString<-ee$Geometry$LineString(list(c(-35, -10), c(35, -10), c(35, 10), c(-35, 10)))
lineString
ee.Geometry({
  "type": "LineString",
  "coordinates": [
    [
      [-35.0,
       -10.0
     ],
     [
       35.0,
       -10.0
     ],
     [
       35.0,
       10.0
     ],
     [
       -35.0,
       10.0
     ]
   ]
})
linearRing<-ee$Geometry$LinearRing(list(c(-35, -10), c(35, -10), c(35, 10), c(-35, 10), c(-35, -10)))
linearRing
ee.Geometry({
  "type": "LinearRing",
  "coordinates": [
    [
      [-35.0,
       -10.0
     ],
     [
       35.0,
       -10.0
     ],
     [
       35.0,
       10.0
     ],
     [
       -35.0,
       10.0
     ],
     [
       -35.0,
       -10.0
     ]
   ]
})
```

```

rectangle<-ee$Geometry$Rectangle(-40,-20,40,20)
rectangle
ee.Geometry({
  "type": "Polygon",
  "coordinates": [
    [
      [
        [
          [-40.0,
           20.0],
          [
            [-40.0,
             -20.0]
          ],
          [
            [40.0,
             -20.0]
          ],
          [
            [40.0,
             20.0]
          ]
        ]
      ]
    ],
    "evenOdd": true
  }
)
polygon<-ee$Geometry$Polygon(list(c(-5,40),c(65,40),c(65,60),c(-5,60),c(-5,40)))
polygon
ee.Geometry({
  "type": "Polygon",
  "coordinates": [
    [
      [
        [
          [-5.0,
           40.0],
          [
            [65.0,
             40.0]
          ],
          [
            [65.0,
             60.0]
          ],
          [
            [-5.0,
             60.0]
          ],
          [
            [-5.0,
             40.0]
          ]
        ]
      ]
    ],
    "evenOdd": true
  }
)

```

Um objeto `Geometry` individual pode ser constituído por múltiplas geometrias. Para separar uma parte dessa geometria múltipla em seus elementos constituintes podemos usar a função `geometry.geometries()`. Por exemplo.

```

multiPoint<-ee$Geometry$MultiPoint(list(c(-121.68, 39.91), c(-97.38, 40.34)))
geometries<-multiPoint$geometries()
pt1<-geometries$get(0)
pt2<-geometries$get(1)
pt1
ee.ComputedObject({

```

```

    "type": "Invocation",
    "arguments": {
      "list": {
        "type": "Invocation",
        "arguments": {
          "geometry": {
            "type": "MultiPoint",
            "coordinates": [
              [
                [-121.68, 39.91],
                [-97.38, 40.34]
              ]
            ]
          }
        },
        "functionName": "Geometry.geometries"
      },
      "index": 0.0
    },
    "functionName": "List.get"
  )
pt2
ee.ComputedObject({
  "type": "Invocation",
  "arguments": {
    "list": {
      "type": "Invocation",
      "arguments": {
        "geometry": {
          "type": "MultiPoint",
          "coordinates": [
            [
              [-121.68, 39.91],
              [-97.38, 40.34]
            ]
          ]
        }
      },
      "functionName": "Geometry.geometries"
    },
    "index": 1.0
  },
  "functionName": "List.get"
})

```

Planar x Geodésico

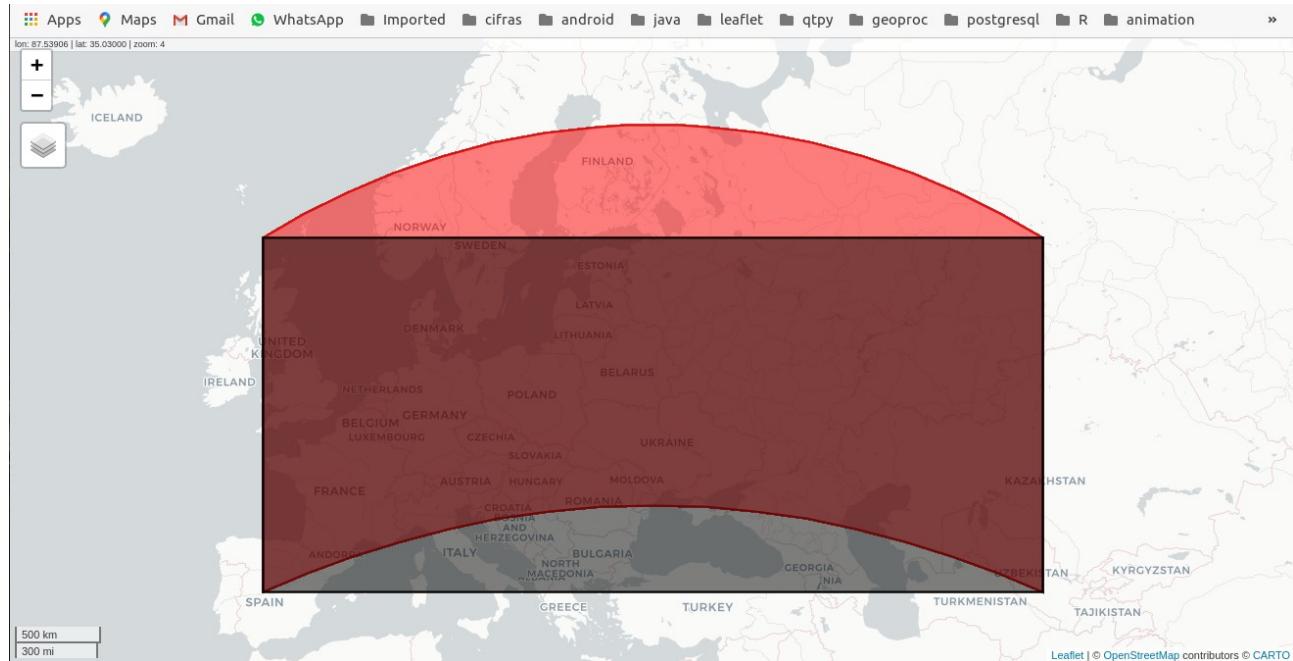
Um objeto `geometry` criado no Earth Engine é geodésico (laterais percorrem o caminho mais curto na superfície esférica da terra) ou planar (laterais percorre o caminho mais curto num plano 2D cartesiano). Nenhuma coordenada planar é adequada para o superfície do planeta como um todo, então a geometria no Earth Engine é por padrão do tipo geodésico. Para criar geometria planar usamos o parâmetro `geodesic` com o valor `FALSE`.

```
planarPolygon<-ee$Geometry(polygon, {}, FALSE)
```

Neste exemplo ilustramos a diferença entre planar (preto) e geodésico (vermelho).

```
polygon <- ee$Geometry$Polygon(list(c(-5, 40), c(65, 40), c(65, 60), c(-5, 60), c(-5, 40)))
planarPolygon <- ee$Geometry(polygon, {}, FALSE)
polygon <- ee$FeatureCollection(polygon)
planarPolygon <- ee$FeatureCollection(planarPolygon)
Map$centerObject(polygon, zoom = 4)
Map$addLayer(polygon, list(color = "FF0000"), "Polígono geodésico") +
Map$addLayer(planarPolygon, list(color = "000000"), "Polígono planar")
```

O Mapa resultante é:



Informações e metadados de ee\$Geometry

Vimos acima com visualizamos os objetos ee\$geometry em mapa, bastante similar ao que fazemos com objetos ee\$image. Agora veremos como acessar e ver informações de metadado associados ao objeto.

Visualizando Usando a função ee_print().

```
polygon <- ee$Geometry$Polygon(list(c(-5, 40), c(65, 40), c(65, 60), c(-5, 60), c(-5, 40)))
ee_print(polygon)
```

Geometry Metadata:

- EPSG (SRID)	: 4326
- proj4string	: +proj=longlat +datum=WGS84 +no_defs
- Geotransform	: 1 0 0 0 1 0
- Geodesic	: TRUE
- WKT	: POLYGON

Para acessar informações individuais sobre um objeto Geometry podemos usar:

```
polygon$area()$getInfo()
[1] 9.918986e+12
polygon$perimeter()$getInfo()
[1] 13979887
polygon$toGeoJSONString()
[1] "{\"type\": \"Polygon\", \"coordinates\": [[[[-5.0, 40.0], [65.0, 40.0], [65.0, 60.0], [-5.0, 60.0], [-5.0, 40.0]]], \"evenOdd\": true}"
polygon$type()$getInfo()
[1] "Polygon"
polygon$coordinates()$getInfo()
[[1]]
[[1]][[1]]
[1] -5 40

[[1]][[2]]
[1] 65 40

[[1]][[3]]
[1] 65 60

[[1]][[4]]
[1] -5 60

[[1]][[5]]
[1] -5 60

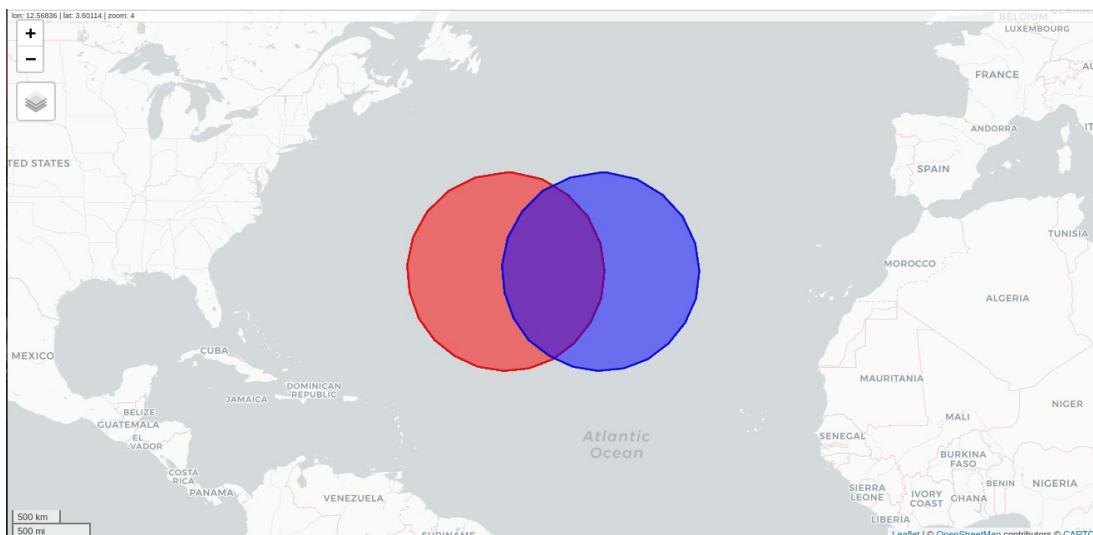
[[1]][[6]]
[1] -5 40

polygon$geodesic()$getInfo()
[1] TRUE
```

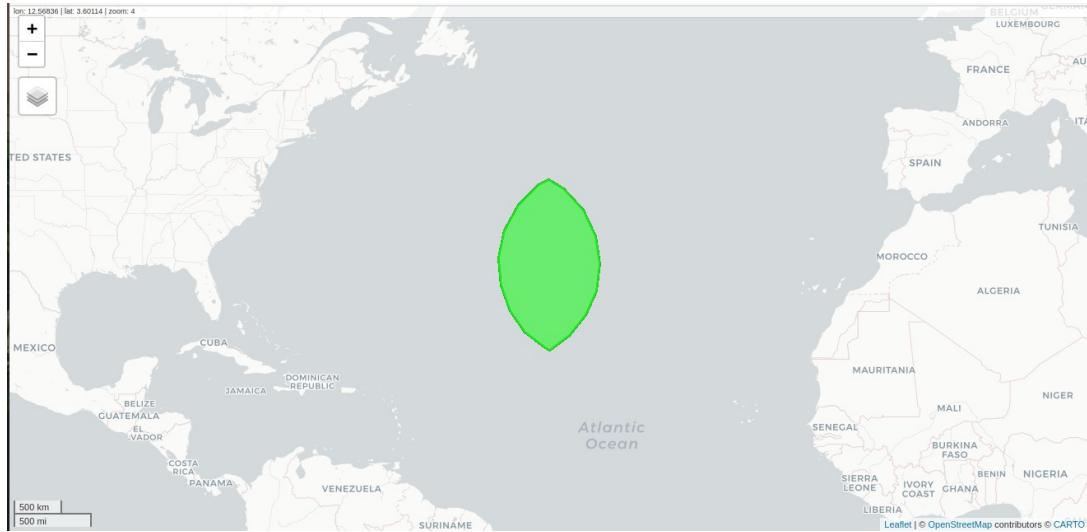
Operações com ee\$Geometry

Objetos Geometry possuem formas de executar operações geoespaciais no Earth Engine. Vamos ver alguma delas usando duas geometrias circulares.

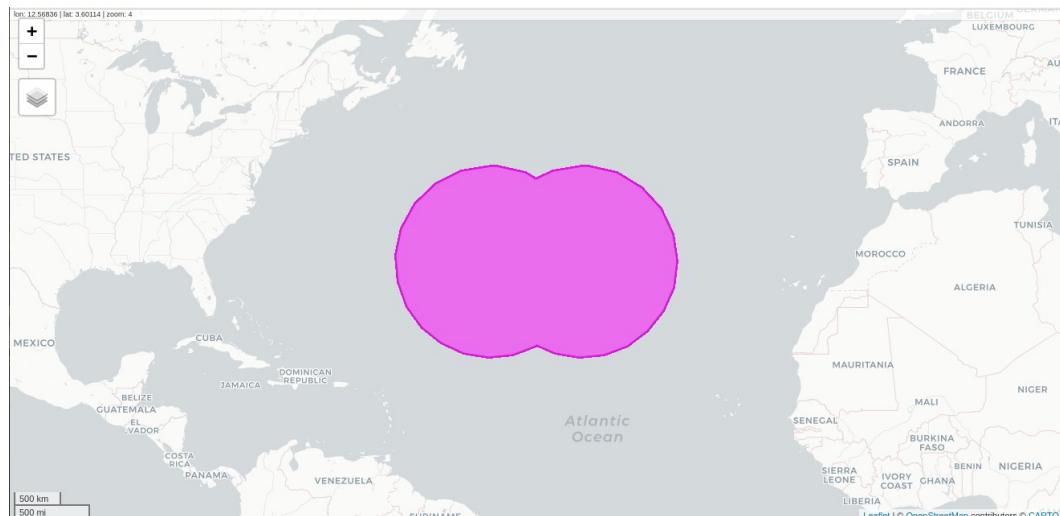
```
poly1 <- ee$Geometry$Point(c(-50, 30))$buffer(1e6)
poly2 <- ee$Geometry$Point(c(-40, 30))$buffer(1e6)
Map$setCenter(-45, 30, 4)
Map$addLayer(poly1, list(color = "FF0000"), "poly1") +
Map$addLayer(poly2, list(color = "0000FF"), "poly2")
```



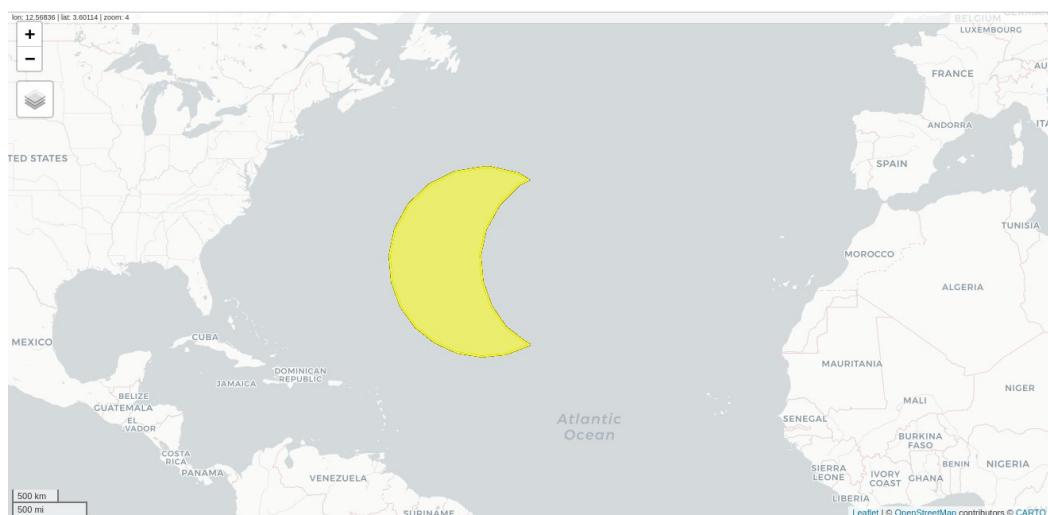
```
intersection <- poly1$intersection(poly2, ee$ErrorMargin(1))
Map$addLayer(intersection, list(color = "00FF00"), "intersecção")
```



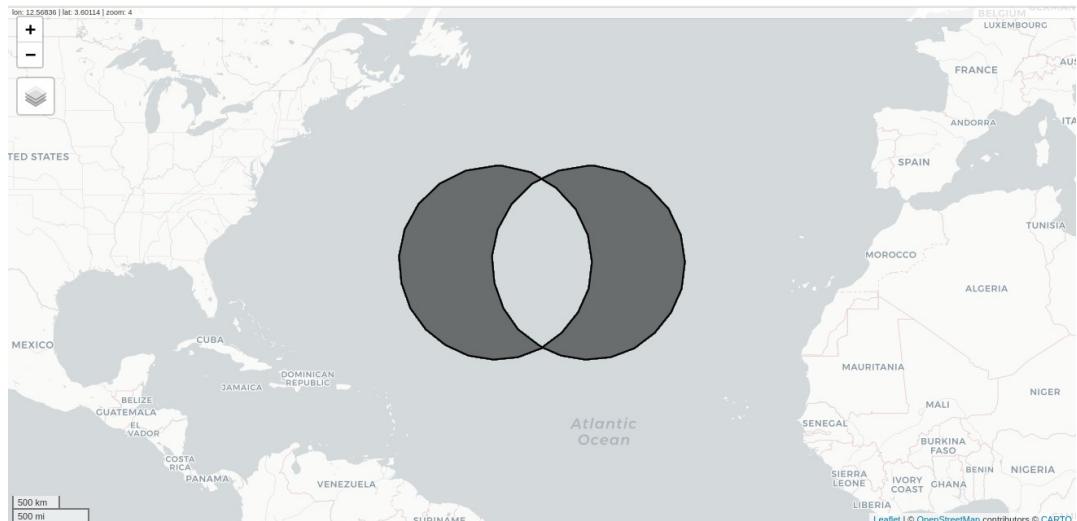
```
union <- poly1$union(poly2, ee$ErrorMargin(1))
Map$addLayer(union, list(color = "FF00FF"), "união")
```



```
diff1 <- poly1$difference(poly2, ee$ErrorMargin(1))
Map$addLayer(diff1, list(color = "FFFF00"), "diferença")
```



```
symDiff <- poly1$symmetricDifference(poly2, ee$ErrorMargin(1))
Map$addLayer(symDiff, list(color = "000000"), "diferença simétrica")
```

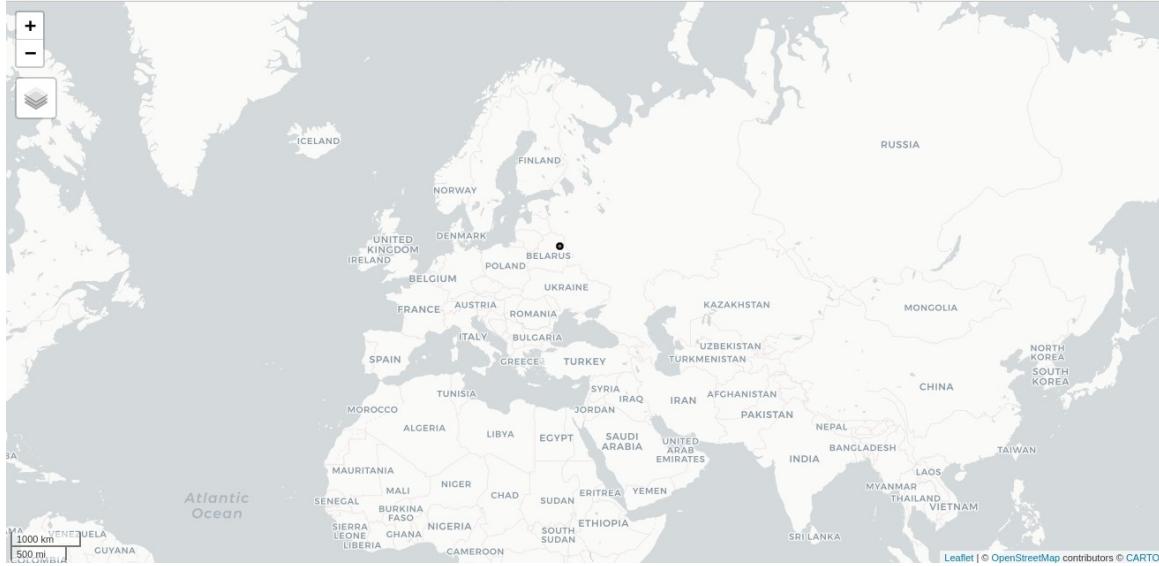


Calculando buffer e centroid de uma Geometria.

```
polygon<-ee$Geometry$Polygon(list(c(-5,40),c(65,40),c(65,60),c(-5,60),c(-5,40)))
buffer<-polygon$buffer(1000000)
centroid<-polygon$centroid()
Map$setCenter(35,50,3)
Map$addLayer(buffer,list(),'buffer')
```



```
Map$addLayer(centroid, list(), 'centroid')
```



ee\$Feature

No Earth Engine um objeto Feature é definido com uma característica GeoJSON. Especificamente, uma característica ou Feature é um objeto composto por uma propriedade geométrica com um objeto Geometry (ou vazio) e uma propriedade property com um dicionário de outras propriedades não geométricas ou atributos.

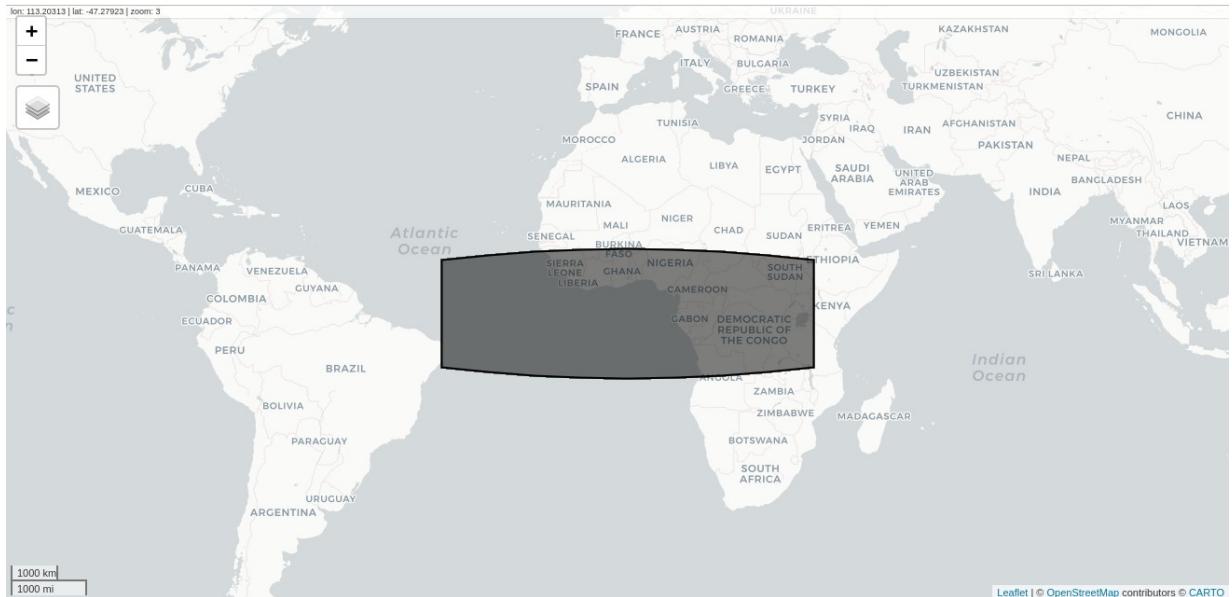
Neste exemplo criamos um objeto ee\$Feature.

```
polygon<-ee$Geometry$Polygon(list(c(-35,-10),c(35,-10),c(35,10),c(-35,10),c(-35,-10)))
polyFeature<-ee$Feature(polygon, list(foo=42, bar='tart'))
ee_print(polyFeature)
```

Earth Engine Feature —

```
Feature Metadata:
- Number of Properties      : 2
Geometry Metadata:
- EPSG (SRID)               : 4326
- proj4string                : +proj=longlat +datum=WGS84 +no_defs
- Geotransform               : 1 0 0 0 1 0
- Geodesic                   : TRUE
- WKT                        : POLYGON
```

E plotando no mapa.



As informações do objeto podem ser vistas usando:

```
polyFeature$info()
$type
[1] "Feature"

$geometry
$geometry$type
[1] "Polygon"
$geometry$coordinates
$geometry$coordinates[[1]]
$geometry$coordinates[[1]][[1]]
[1] -35 -10
$geometry$coordinates[[1]][[2]]
[1] 35 -10
$geometry$coordinates[[1]][[3]]
[1] 35 10
$geometry$coordinates[[1]][[4]]
[1] -35 10
$geometry$coordinates[[1]][[5]]
[1] -35 -10

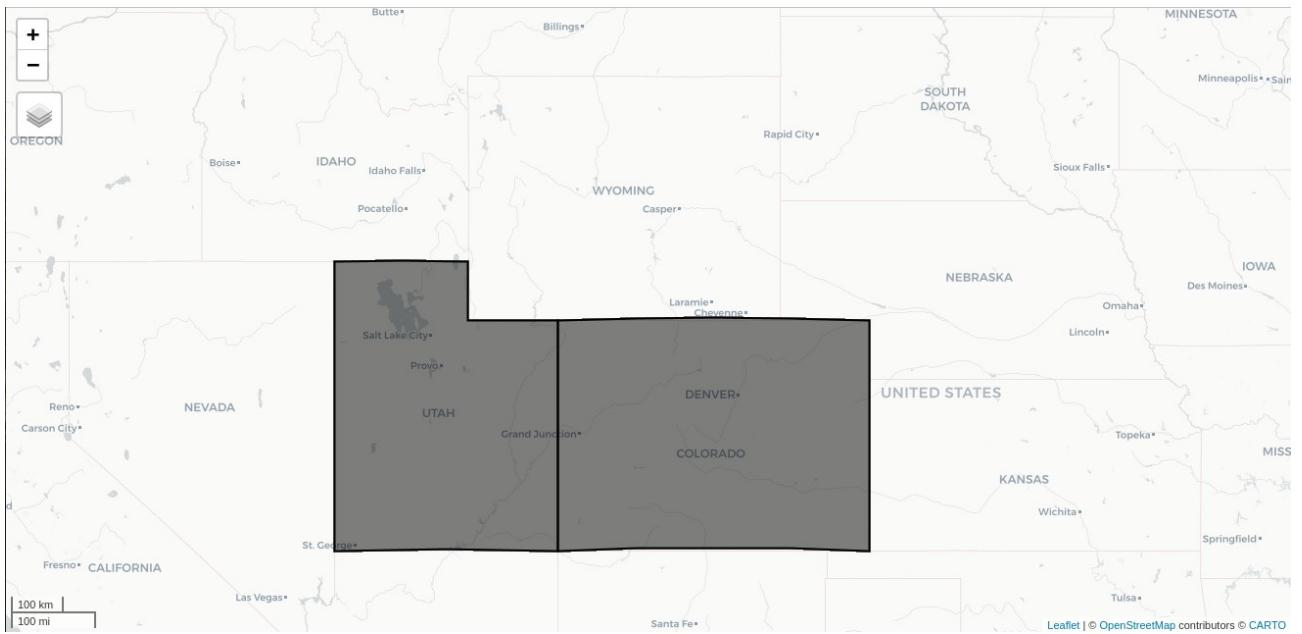
$properties
$properties$bar
[1] "tart"

$properties$foo
[1] 42
```

ee\$FeatureCollection

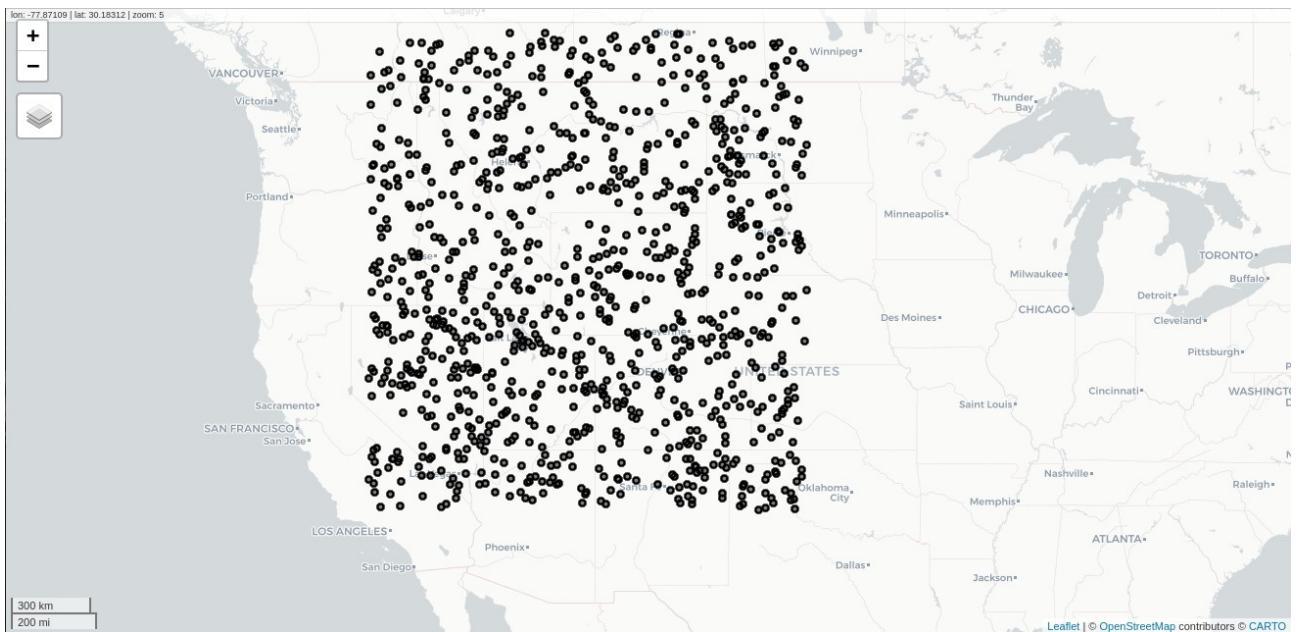
Mostraremos agora como podemos criar uma Feature Collection e vamos visualizar a mesma no mapa.

```
fc <- ee$FeatureCollection(list(
ee$Feature(ee$Geometry$Polygon(list(c(-109.05, 41),c(-109.05,37),c(-102.05,37),c(-102.05,41))),list(name="Colorado",fill=1)),
ee$Feature(ee$Geometry$Polygon(list(c(-114.05,37.0),c(-109.05,37.0),c(-109.05,41.0),c(-111.05,41.0),c(-111.05,42.0),c(-114.05,42.0))),list(name="Utah",fill=2))
))
Map$setCenter(-107,41,6)
Map$addLayer(eeObject=fc,visParams=list(palette=c("000000","FF0000","00FF00","0000FF"),max=3,opacity=0.5),name="Colorado & Utah")
```



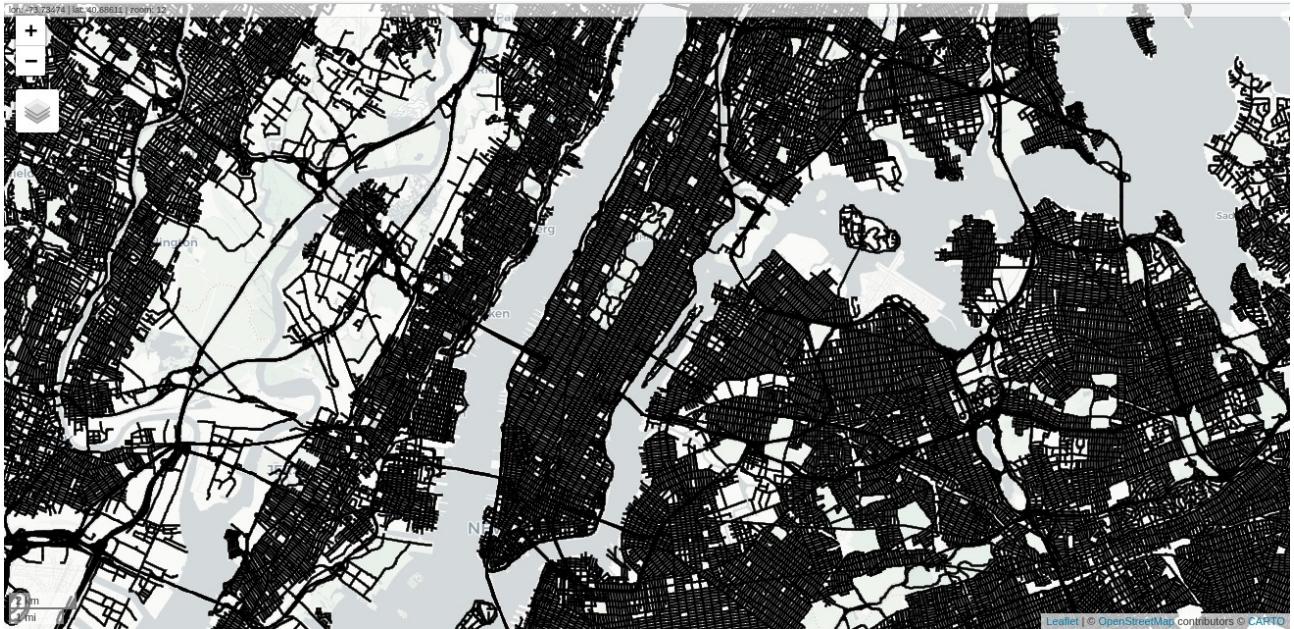
Agora criando uma coleção de features usando pontos aleatórios.

```
region<-ee$Geometry$Rectangle(-119.224, 34.669, -99.536, 50.064)
randomPoints<-ee$FeatureCollection$randomPoints(region)
Map$setCenter(-107, 41, 5)
Map$addLayer(randomPoints, list(), 'Pontos Aleatórios')
```



Podemos criar uma coleção de features com base em dados do Earth Engine.

```
fc<-ee$FeatureCollection('TIGER/2016/Roads')
Map$setCenter(-73.9596, 40.7688, 12)
Map$addLayer(fc, list(), 'Ruas NY Censo')
```

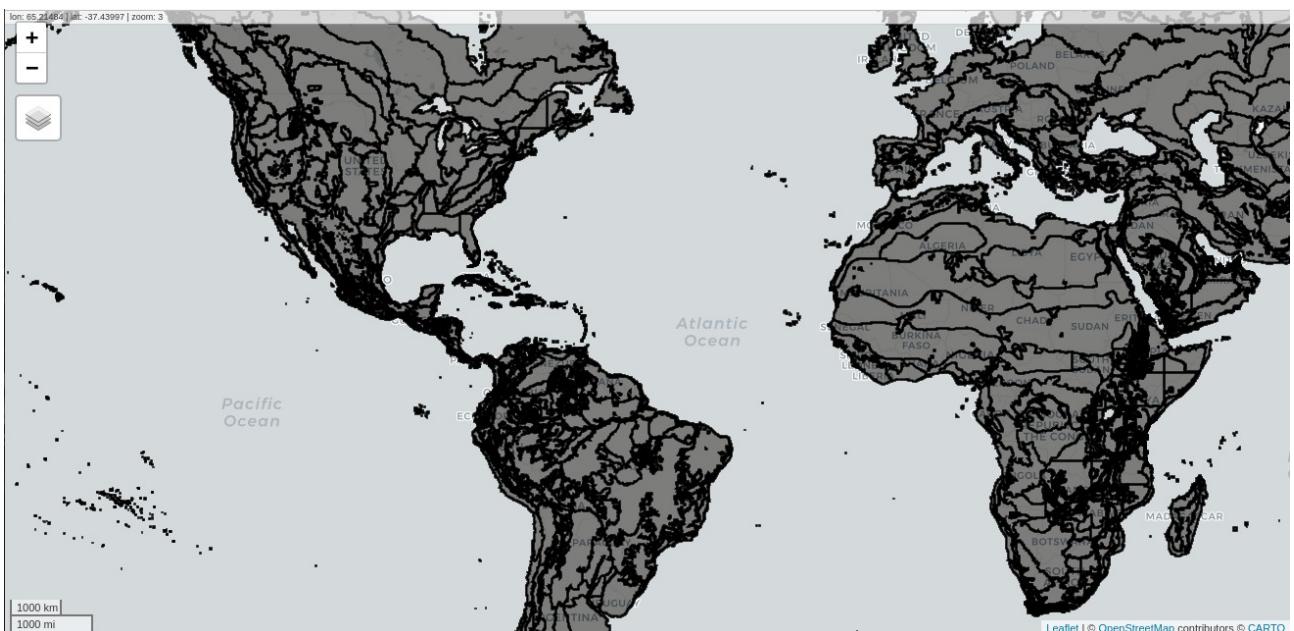


Visualização de ee\$FeatureCollection

Vamos ver agora com um pouco mais de detalhes como trabalhar com a visualização de um objeto ee\$FeatureCollection.

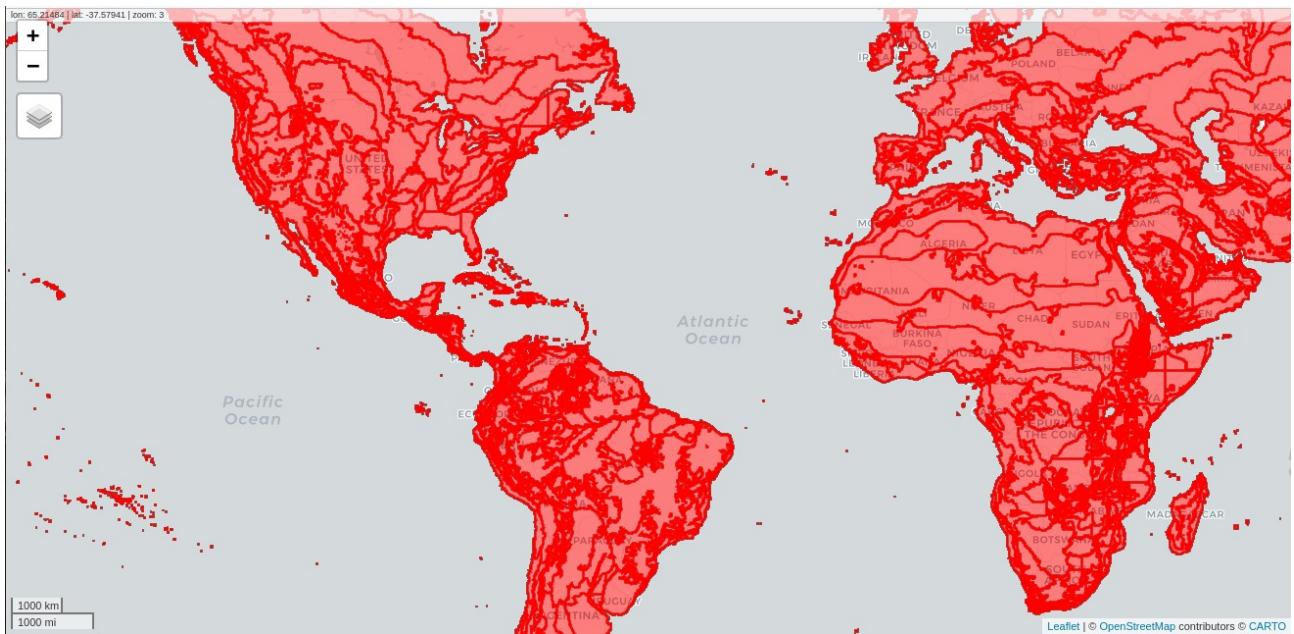
Primeiro criando o feature e usando a visualização padrão.

```
Map$setCenter(-50,15,3)
ecoregions<-ee$FeatureCollection('RESOLVE/ECOREGIONS/2017')
Map$addLayer(ecoregions, list(), 'Padrão')
```



Usando parâmetros de visualização

```
Map$addLayer(ecoregions, list(color='FF0000'), 'colorido')
```



E usando a função draw.

```
Map$addLayer(ecoregions$draw(color='green', strokeWidth=5), list(), 'desenhado')
```



Podemos de maneira eficiente categorizar o que vemos usando com base nas propriedades da coleção de Features. Isso é feito usando a função `paint`. Inicialmente vamos usar valores fixos para as bordas.

```
empty<-ee$Image()$byte()  
outline<-empty$paint(featureCollection=ecoregions,color=1,width=3)  
Map$addLayer(outline, list(palette='#FF0000'), 'bordas')
```



Agora usando o parâmetro (propriedade) 'BIOME_NUM' para a cor da borda e criando uma paleta de cor RGB.

```
outlines<-empty$paint(featureCollection=ecoregions,color='BIOME_NUM',width=2)  
palette<-c('#FF0000', '#00FF00', '#0000FF')  
Map$addLayer(outlines, list(palette=palette, max=14), 'Bordas de cor diferente')
```



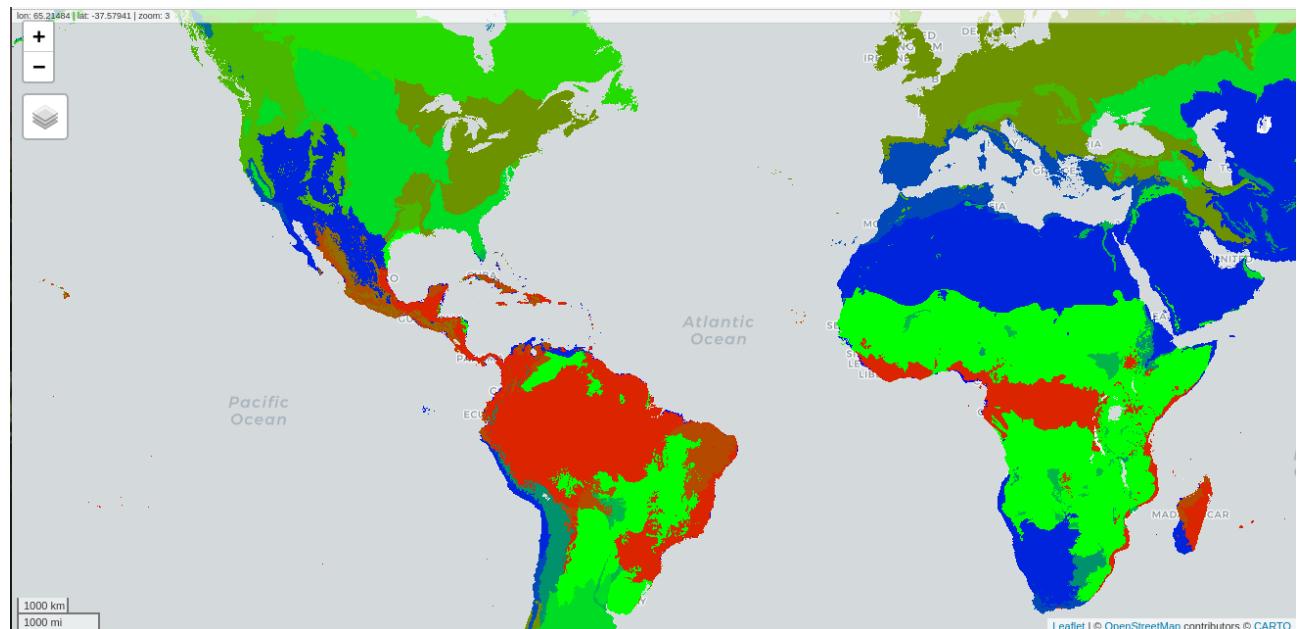
E usando a propriedade 'NNH' para definir a espessura da borda.

```
outlines<-  
empty$paint(featureCollection=ecoregions,color='BIOME_NUM',width='NNH')  
Map$addLayer(outlines, list(palette=palette, max=14), 'Bordas e cor e espessura  
diferentes')
```



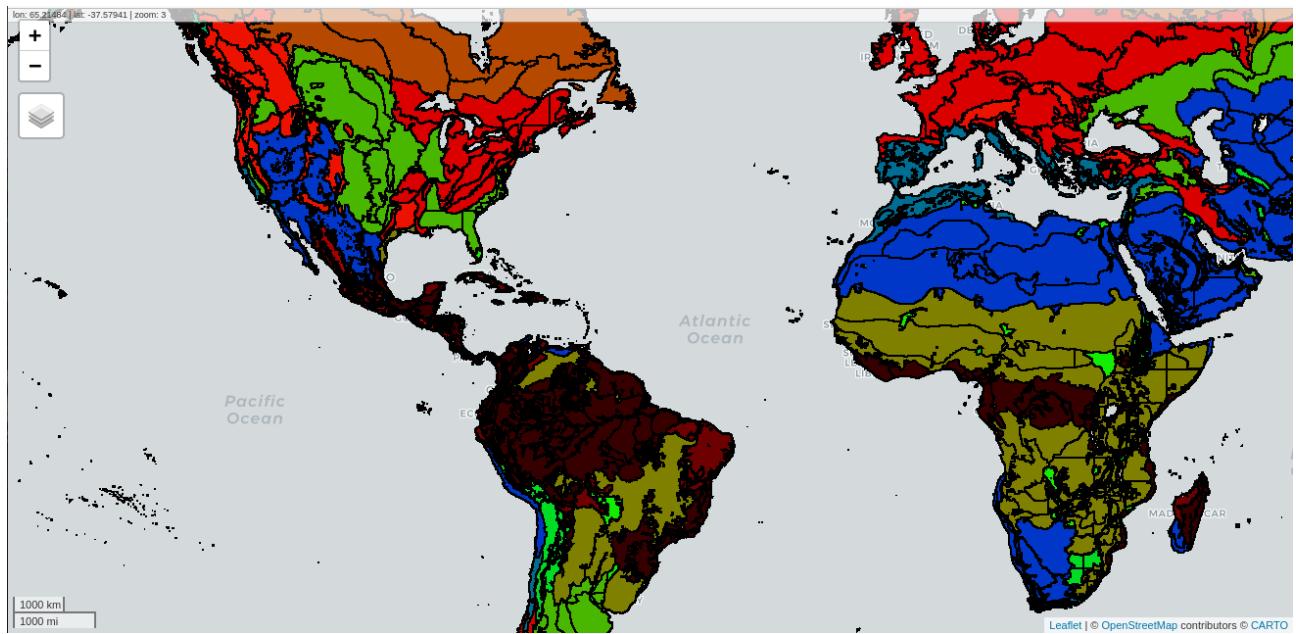
Podemos associar ao preenchimento usando.

```
fills<-empty$paint(featureCollection=ecoregions,color='BIOME_NUM')  
Map$addLayer(fills, list(palette=palette, max=14), 'Preenchimento colorido')
```



Para finalizar, usamos bordas e preenchimento.

```
filledOutlines<-empty$paint(ecoregions, 'BIOME_NUM')$paint(ecoregions, 0, 2)
Map$addLayer(filledOutlines, list(palette= c('#000000',palette), max=14), 'Bordas e preenchimentos')
```



Informações e Metadados de ee\$FeatureCollection

Vamos agora ver como obter informações sobre a coleção de feature.

Primeiramente usando ee_print().

```
sheds<-ee$FeatureCollection('USGS/WBD/2017/HUC06')
$filterBounds(ee$Geometry$Rectangle(-127.18, 19.39, -62.75, 51.29))
$map(function(feature){
  num<-ee$Number$parse(feature$get('areasqkm'))
  return (feature$set('areasqkm', num))
})
ee_print(sheds)
```

Earth Engine FeatureCollection —

```
FeatureCollection Metadata:
- Class : ee$FeatureCollection
- Number of Features : 336
- Number of Properties : 14
Feature Metadata:
- Number of Properties : 14
Geometry Metadata (f_index = 0):
- EPSG (SRID) : 4326
- proj4string : +proj=longlat +datum=WGS84 +no_defs
- Geotransform : 1 0 0 0 1 0
- Geodesic : TRUE
- WKT : POLYGON
```

```
Map$setCenter(-90,35,4)
Map$addLayer(sheds, list(), 'watersheds')
```



Obtendo informações específicas sobre a FeatureCollection

```
sheds$size()$getInfo()
[1] 336
sheds$aggregate_stats('areasqkm')$getInfo()
$type
[1] "DataDictionary"
$values
$values$max
[1] 123604.1
$values$mean
[1] 25513.32
$values$min
[1] 601.89
$values$sample_sd
[1] 17082.47
$values$sample_var
[1] 291810677
$values$sum
[1] 8572476
$values$sum_sq
[1] 316468933433
$values$total_count
[1] 336
$values$total_sd
[1] 17057.03
$values$total_var
[1] 290942193
$values$valid_count
[1] 336
$values$weight_sum
[1] 336
$values$weighted_sum
[1] 8572476
```

Para obter informações sobre as colunas de dados de atributos de um feature dentro de um FeatureCollection podemos usar.

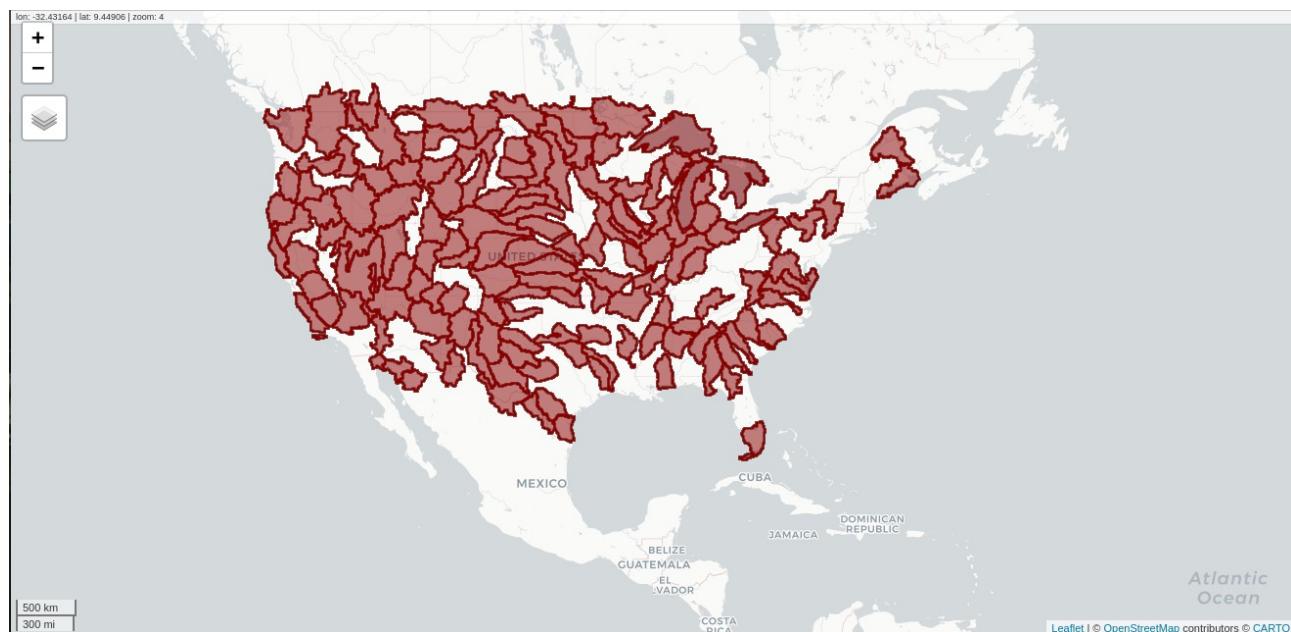
```
wdpa<-ee$FeatureCollection("WCMC/WDPA/current/points")
colunas<-wdpa$first()$getInfo()
names(colunas$properties)
[1] "DESIG"          "DESIG_ENG"       "DESIG_TYPE"      "GOV_TYPE"       "INT_CRIT"
[6] "ISO3"           "IUCN_CAT"        "MANG_AUTH"      "MANG_PLAN"      "MARINE"
[11] "METADATAID"    "NAME"          "NO_TAKE"        "NO_TK_AREA"    "ORIG_NAME"
[16] "OWN_TYPE"       "PARENT_ISO"     "PA_DEF"         "REP_AREA"      "REP_M_AREA"
[21] "STATUS"         "STATUS_YR"       "SUB_LOC"        "VERIF"         "WDPAID"
[26] "WDPA_PID"
```

Filtrando ee\$FeatureCollection

Vamos agora abordar como é feita a filtragem de uma coleção Feature. Ela se dá de forma semelhante ao que fizemos com filtragem de ImageCollection.

Neste exemplo vamos carregar os dados das bacias hidrográficas e filtrar dentro de uma área equivalente ao EUA continental. Checamos o número total de bacias e em seguida filtramos para mostrar somente as bacias maiores que 25000 km quadrados.

```
sheds<-ee$FeatureCollection('USGS/WBD/2017/HUC06')$map(function(feature){  
  num<-ee$Number$pparse(feature$get('areasqkm'))  
  return (feature$set('areasqkm', num))})  
continentalUS<-ee$Geometry$Rectangle(-127.18, 19.39, -62.75, 51.29)  
filtered<-sheds$filterBounds(continentalUS)  
filtered$count()$getInfo()  
[1] 336  
largeSheds<-filtered$filter(ee$Filter$gt('areasqkm', 25000))  
largeSheds$count()$getInfo()  
[1] 140  
Map$setCenter(-90,35,4)  
Map$addLayer(largeSheds, list(color='#880000'), 'watersheds')
```



Mapeando um ee\$FeatureCollection

Vamos agora mostrar como podemos alterar todos os features de uma featureCollection usando map. Da mesma forma que usamos com ImageCollection, map é usado com FeatureCollection passando um objeto feature como parâmetro ao invés de um objeto image.

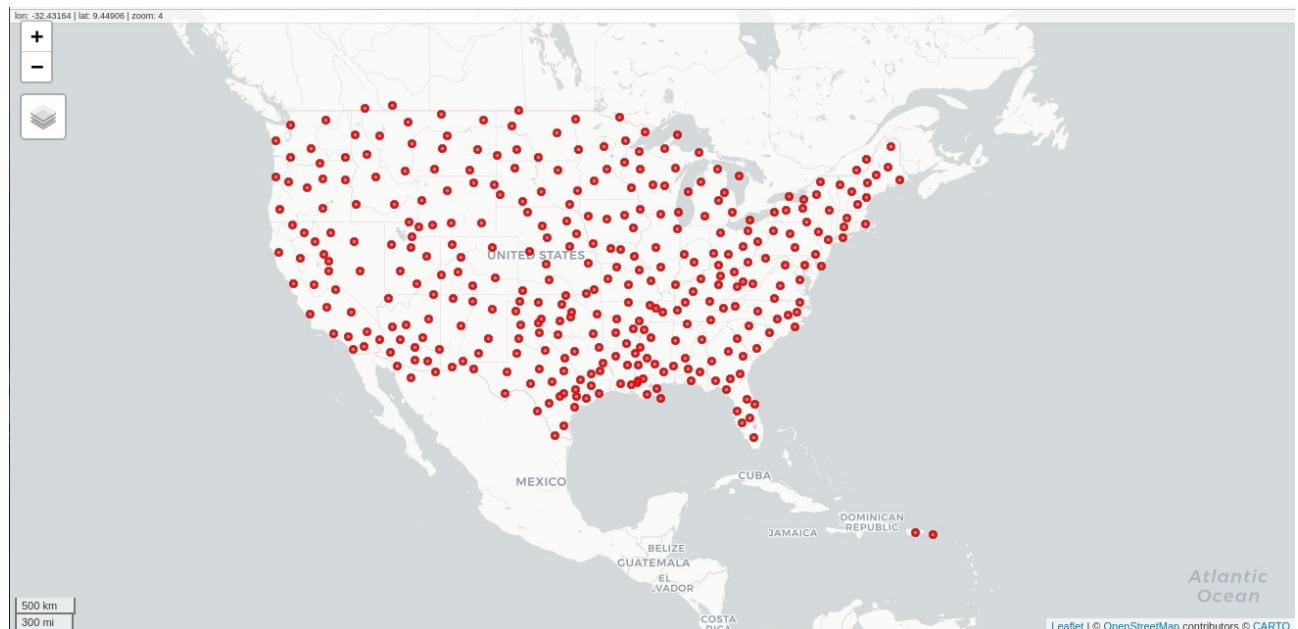
Vejamos como adicionar uma coluna.

```
sheds<-ee$FeatureCollection('USGS/WBD/2017/HUC06')
colunas<-sheds$first()$getInfo()
colunas$properties
$areaacres
[1] "17935746.85"
$areasqkm
[1] "72583.46"
$gnis_id
[1] ""
$huc6
[1] "190301"
$loaddate
[1] "20120611075458"
$metasource
[1] ""
$name
[1] "Aleutian Islands"
$shape_area
[1] "9.9308571861444"
$shape_leng
[1] "81.5083456556245"
$sourcedata
[1] ""
$sourcefeat
[1] ""
$sourceorig
[1] ""
$states
[1] "AK"
$tnmid
[1] "{7D8E9EC1-6A0D-4FED-8063-94688560AC75}"
addArea<-function(feature){
  return  (feature$set(list(areaHa=  feature$geometry()$area()$divide(100  *
100))))
}
areaAdded<-sheds$map(addArea)
colunas2<-areaAdded$first()$getInfo()
colunas2$properties
$areaHa
[1] 7227401
$areaacres
[1] "17935746.85"
$areasqkm
[1] "72583.46"
$gnis_id
[1] ""
$huc6
[1] "190301"
$loaddate
[1] "20120611075458"
$metasource
[1] ""
$name
[1] "Aleutian Islands"
$shape_area
[1] "9.9308571861444"
$shape_leng
[1] "81.5083456556245"
$sourcedata
[1] ""
$sourcefeat
[1] ""
$sourceorig
[1] ""
$states
[1] "AK"
$tnmid
[1] "{7D8E9EC1-6A0D-4FED-8063-94688560AC75}"
```

No exemplo acima criamos uma nova propriedade (atributo) baseado na geometria do feature. Novos atributos podem também ser criados usando computação com parâmetros já presentes.

Uma nova FeatureCollection pode ser inteiramente recriada usando map(). O exemplo abaixo converte o watersheds em centroides e seleciona somente algumas propriedades(atributos).

```
getCentroid<-function(feature) {  
  keepProperties = list('name', 'huc6', 'tnmid', 'areasqkm')  
  centroid = feature$geometry()$centroid()  
  return (ee$Feature(centroid)$copyProperties(feature, keepProperties))  
}  
centroids<- sheds$map(getCentroid)  
Map$addLayer(centroids, list(color='FF0000'), 'centroides')
```



E checando as colunas criadas na coleção centroid.

```
colunas<-centroids$first()$getInfo()  
names(colunas$properties)  
[1] "areasqkm" "huc6"      "name"       "tnmid"  
colunas$properties  
$areasqkm  
[1] "72583.46"  
$huc6  
[1] "190301"  
$name  
[1] "Aleutian Islands"  
$tnmid  
[1] "{7D8E9EC1-6A0D-4FED-8063-94688560AC75}"
```

No próximo volume finalizaremos usando ee\$Reduce e ee\$Join
Fortaleza 16 de julho de 2020.