Earth Engine Big Data Structures

Images and Features

Renee Johnston / September 16 2019

bit.ly/2mi5T0q

Introductions!

Who am I?

// Renee Johnston (reneejohnston@google.com)
// SWE on the Earth Engine team for 3 years
// Now on the Environmental Insights "sibling team"
// Still use Earth Engine all the time.

Who are these other Googlers?

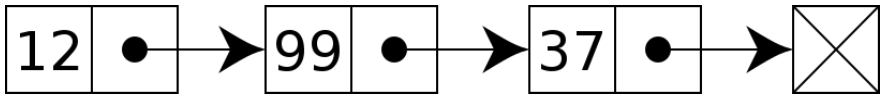
// (Our friendly TAs!)

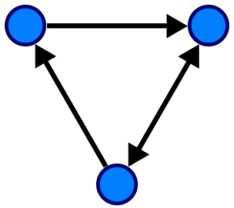
Who are you?

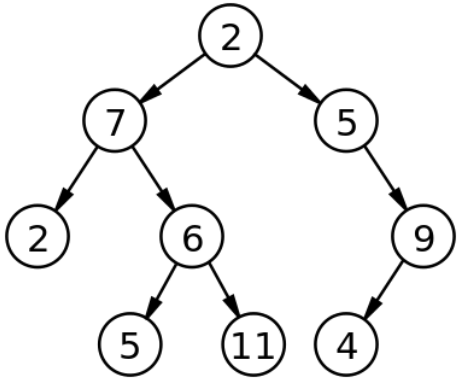
An overview of this course

What are "Data Structures"?

- A fundamental course in an academic Computer Science program
- "A specialized format for organizing, processing, retrieving and storing data"
- Examples: arrays, hash tables (dictionaries), linked lists, graphs, trees.



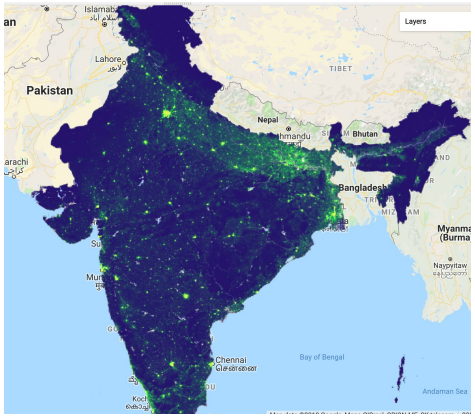




What are the primary "data structures" in Earth Engine?

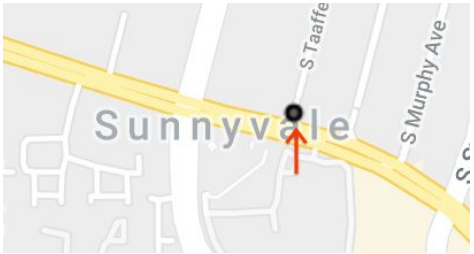
Images

```
ee.Image('WorldPop/GP/100m/pop/IND_2015');
```



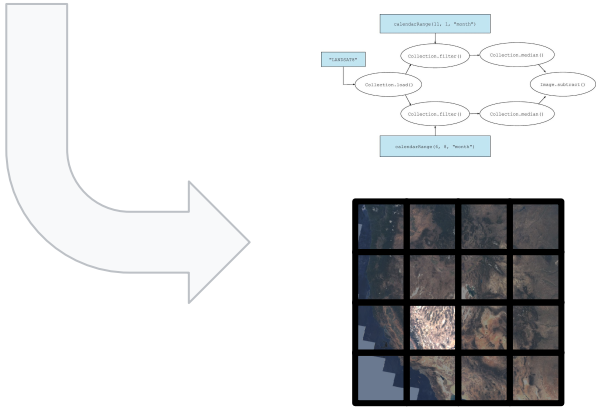
Features

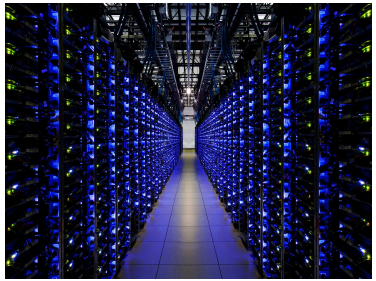
```
var sunnyvale = ee.Feature(
        ee.Geometry.Point([-122.035, 37.369]),
        {"County": "Santa Clara"}
        );
```

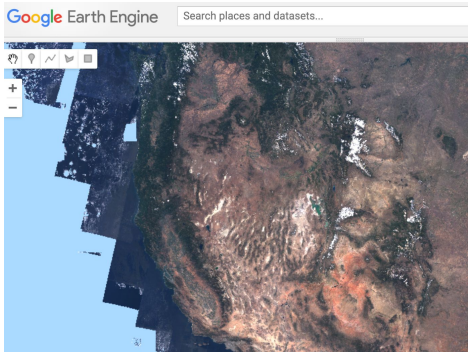


Why "Earth Engine **BIG** Data Structures"?

```
collection = ee.ImageCollection("LANDSAT8")
winter = collection.filter(ee.Filter.calendarRange(11, 1, "month"))
summer = collection.filter(ee.Filter.calendarRange(6, 8, "month"))
diff = summer.median().subtract(winter.median())
```







The structure of this course

First Hour-ish:
Lecture diving into the conceptual underpinnings of "Images" and "Features" in EE.

Feel free to follow along with the slides. (bit.ly/2mi5T0q)

Remaining time:
(Optional) hands-on, self paced lab. (Link at the end of slide deck.)

You're welcome to collaborate and ask for help from me or the TAs.

But also feel free to leave to get to the front of the snack line. :)

Attribution

- This course borrows heavily from resources on the very useful [Earth Engine EDU site]

Especially:

- Nick Clinton's "Introductory Remote Sensing Code Labs"
- Ran Goldblatt's "Introductory Remote Sensing Lectures"

Also:


Remote Sensing of Environment 202 (2017) 18–27

Contents lists available at ScienceDirect

Remote Sensing of Environment

journal homepage: www.elsevier.com/locate/rse

Google Earth Engine: Planetary-scale geospatial analysis for everyone

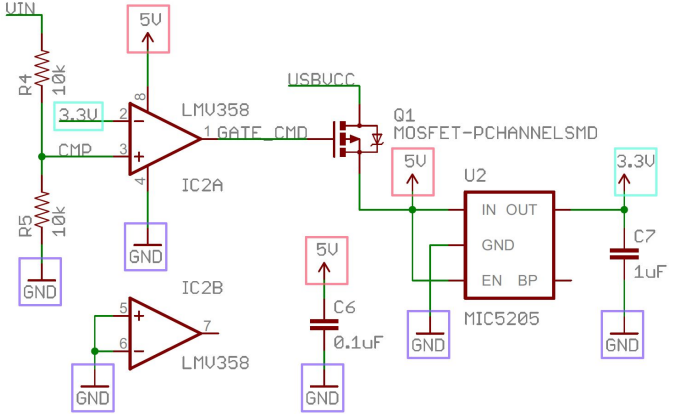
Noel Gorelick [a,*], Matt Hancher [b], Mike Dixon [b], Simon Ilyushchenko [b], David Thau [b], Rebecca Moore [b]

https://www.sciencedirect.com/science/article/pii/S0034425717302900

Earth Engine Images

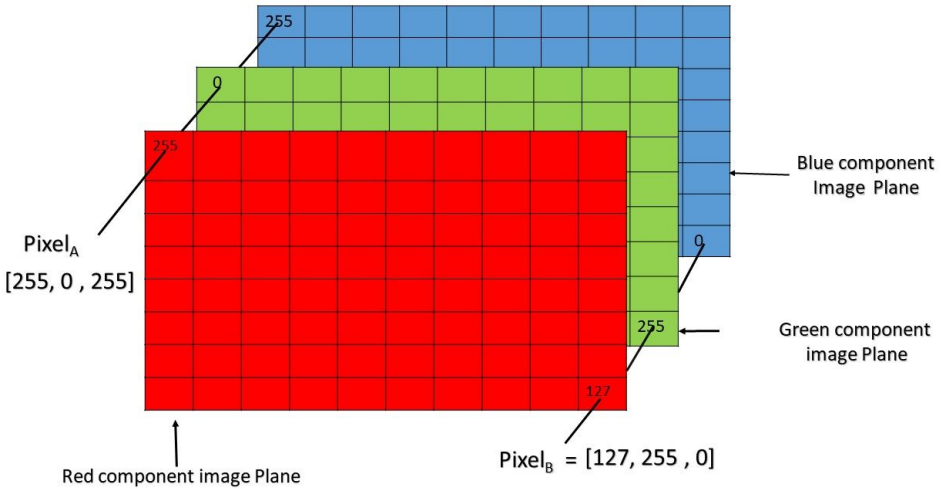
What is an image?

"A representation of the external form of a person or thing"

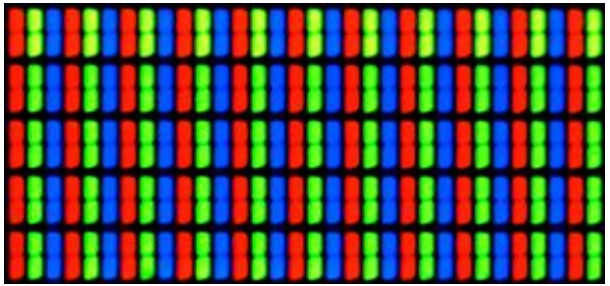


What is an image on a computer?

- A Digital Image - sometimes called a raster - is a "block of pixels"
- Each pixel has a **position**, measured with respect to the axes of some coordinate reference system (CRS)



Pixel of an RGB image are formed from the corresponding pixel of the three component images

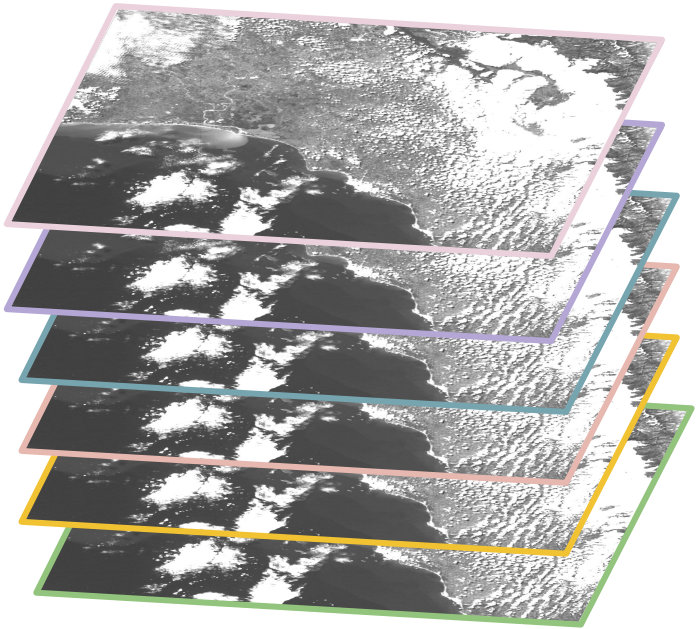


Close up of a typical LCD screen

What is an image in Earth Engine?

Each image has

- **Bands**
  - 2D grid of pixels with a:
    - Name
    - CRS/Projection
    - Pixel Scale/Resolution
- **Properties**, including:
  - Date, Bounding-box, unique ID



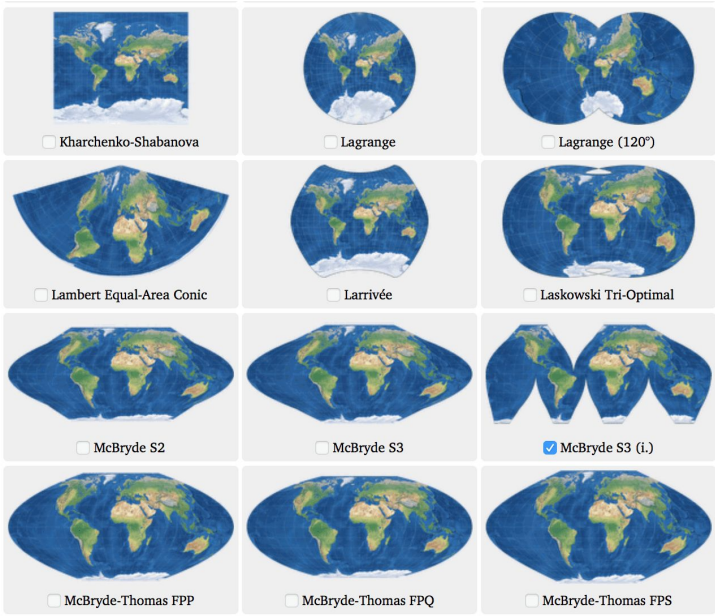
An Image with 6 bands

Defining terms: **Projection and Scale**

Each band in EE has its own Projection.

Components of an EE Projection:

- **CRS**: coordinate reference system(ex: EPSG:4326)
- **Scale**: <u>resolution</u> of the projection to work in. (ex "30 meters per pixel", "1 arc degree per pixel")
- **CRS Transform**: linear mapping from 2D pixel coordinates to 2D projected coordinates. Preserves distances



Some projections

Quick review: Images in the Code Editor

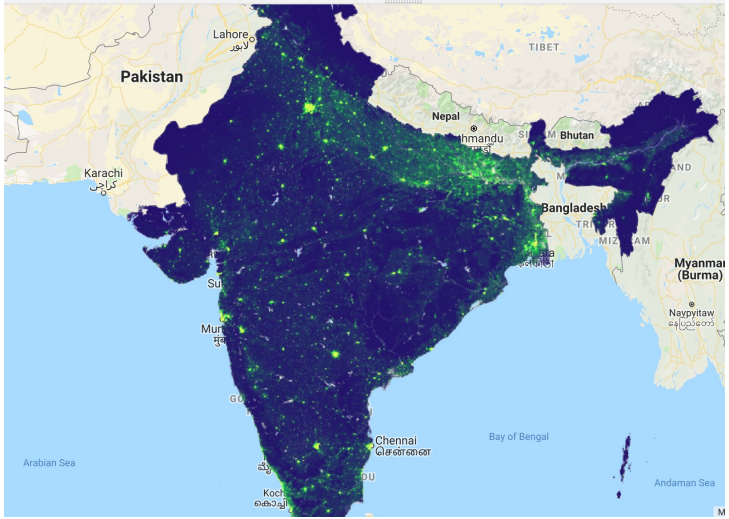
Link 3cb6ff1057... | Get Link | Save | Run | Reset

```
1  // Get the 2015 India Worldpop Image
2  var wpIndia = ee.Image('WorldPop/GP/100m/pop/IND_2015');
3
4  // Print to view metadata
5  print(wpIndia);
6
7
8
9
10
11
12 // Display India WP data on a map
13 var vis = {
14     min: 0.0,
15     max: 50.0,
16     palette: ['24126c', '1fff4f', 'd4ff50'],
17   };
18 Map.addLayer(wpIndia, vis)
19
```

Inspector | Console | Tasks

Use print(...) to write to this console.

```
Image WorldPop/GP/100m/pop/IND_2015 (1 band)
  type: Image
  id: WorldPop/GP/100m/pop/IND_2015
  version: 1565315782859720
  bands: List (1 element)
    0: "population", float, EPSG:4326, 35075x34497 px
  properties: Object (7 properties)
    country: IND
    system:asset_size: 1861467367
    system:footprint: LinearRing, 20 vertices
    system:index: IND_2015
    system:time_end: 1451606400000
    system:time_start: 1420070400000
    year: 2015
```



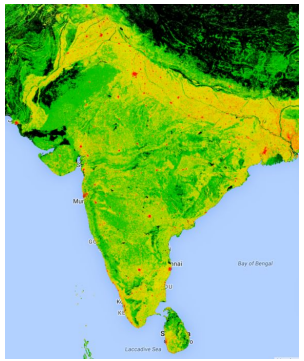
EE Bands: Not Just for Satellite Imagery

EE Bands:

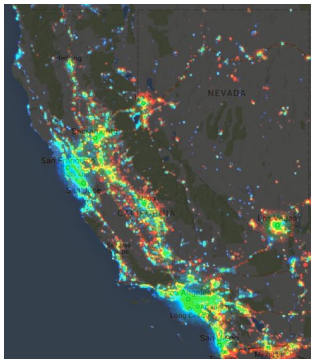
Similar to the Remote Sensing concept of a "band or channel"

But the 2D grid of pixels in an EE Band can represent anything:

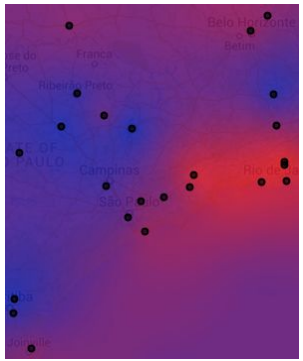
- Ex: Population, elevation, interpolated weather stations



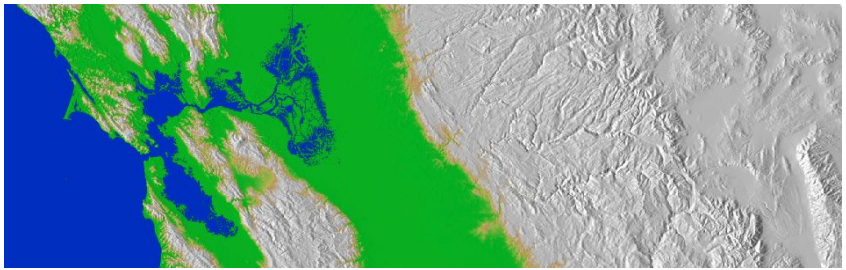
Analyzing population density.



Analyzing night lights trend.



Interpolating weather stations.



Analyzing terrain data.

Other types of EE Images

ee.Image.constant(value)

Generates an image containing a constant value everywhere.

ee.Image.random(*seed*)

Generates a uniform random number at each pixel location, in the range of 0 to 1.

ee.Image.pixelArea()

Generate an image in which the value of each pixel is the area of that pixel in square meters.

ee.Image.pixelLonLat()

Creates an image with two bands named 'longitude' and 'latitude', containing the longitude and latitude at each pixel, in degrees.

EE Images: Behind the scenes

How does Earth Engine:

- **Upload** ("ingest") images?
- **Store** images?
- **Retrieve** images?
- **Compute** on images?

Why is this important to know?

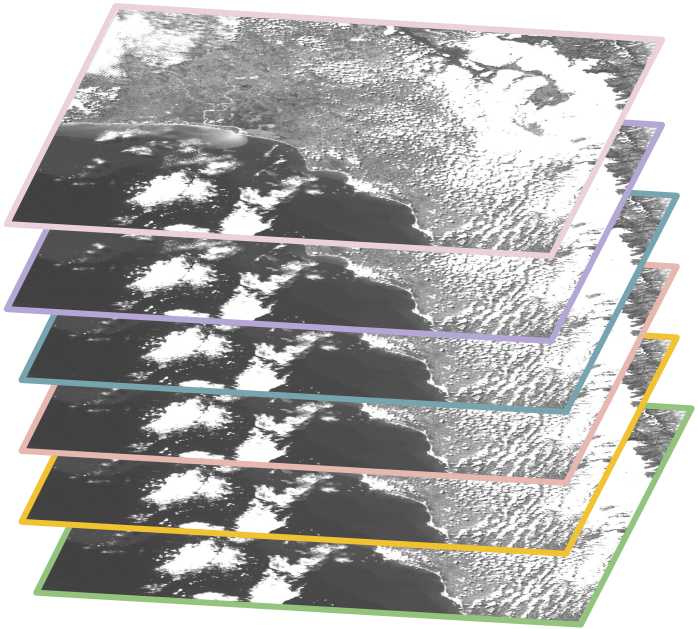


Image **Upload and Ingestion**

Landsat & Sentinel 2
10-30m, 14-day

MODIS
250m daily

Sentinel 1
Radar

Terrain & Land Cover

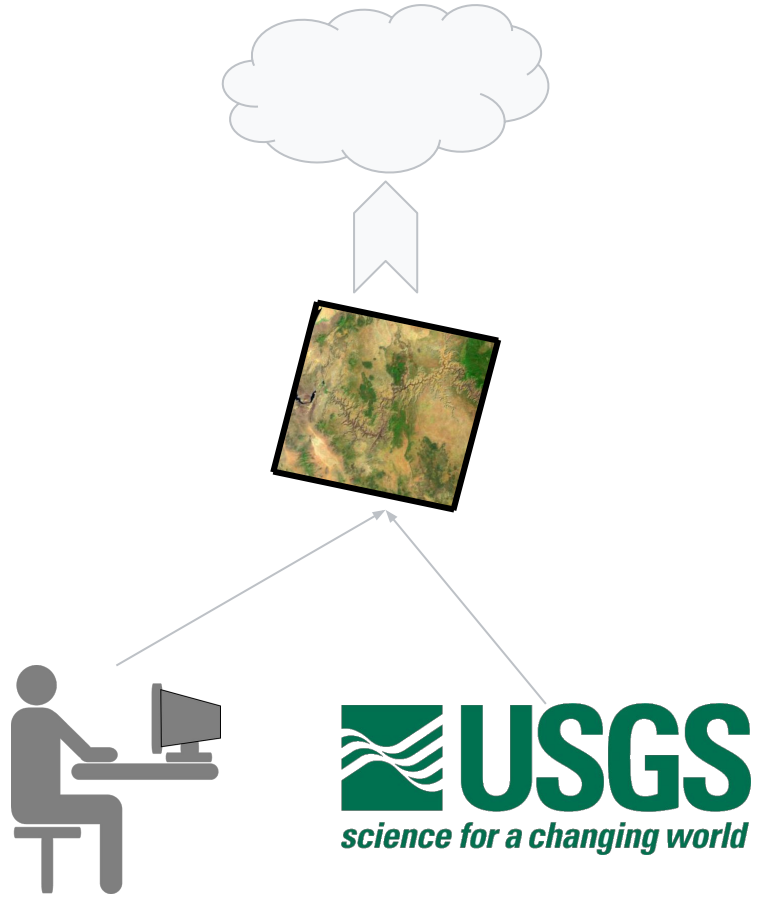
Weather & Climate
NOAA NCEP, OMI, ...

EE uploads **over 6000** scenes into the data catalog **every day**

All processed and stored in a EE-specific format

Ingesting EE Images

1. Upload scene to Google datacenter

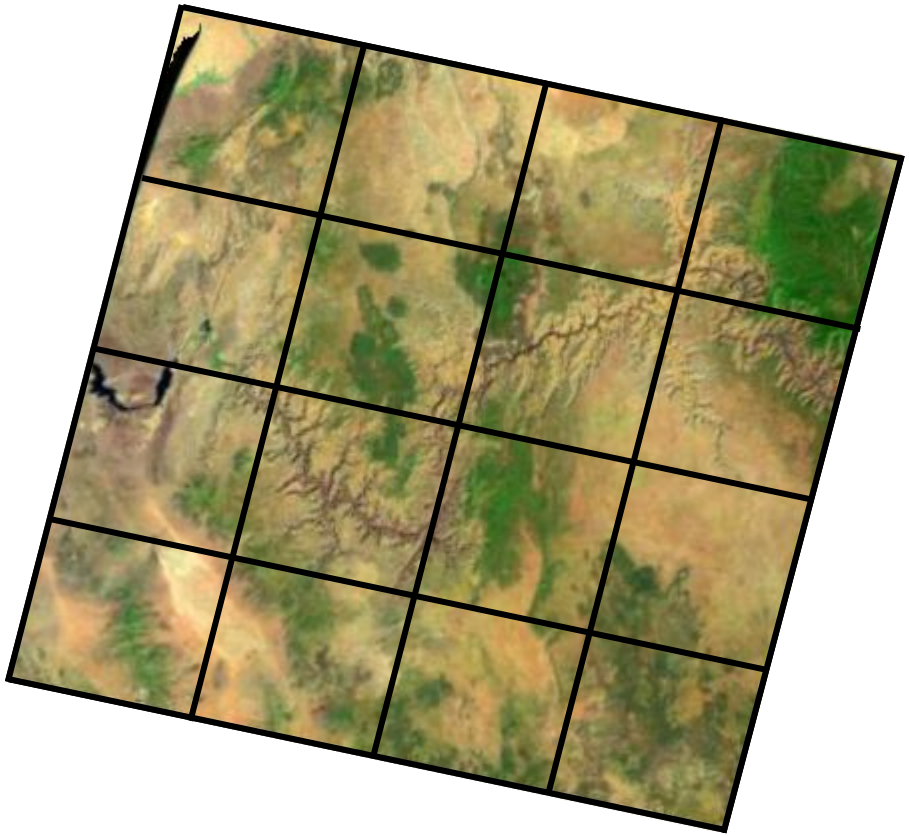


Ingesting EE Images

1. Upload scene to Google datacenter
2. Leave scene in original projection and resolution

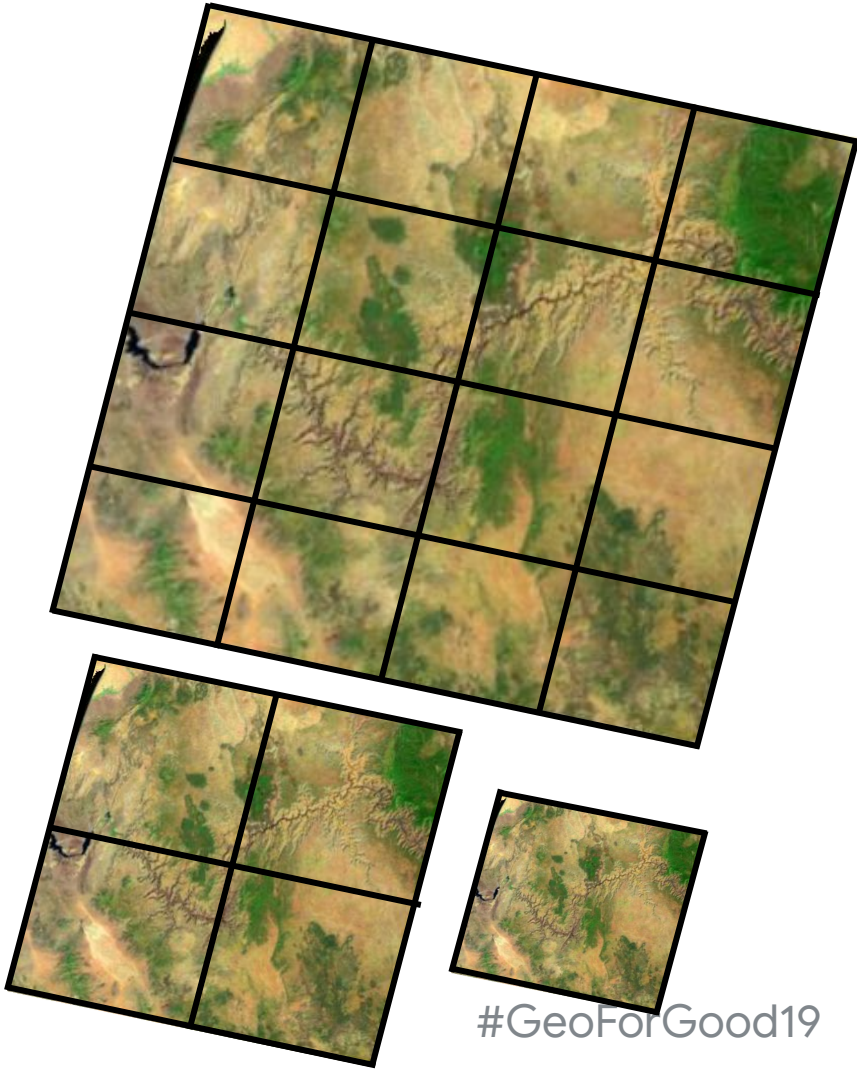
Ingesting EE Images

1. Upload input scene to Google
2. Leave scene in original projection and resolution
3. Break the image into 256 x 256 tiles



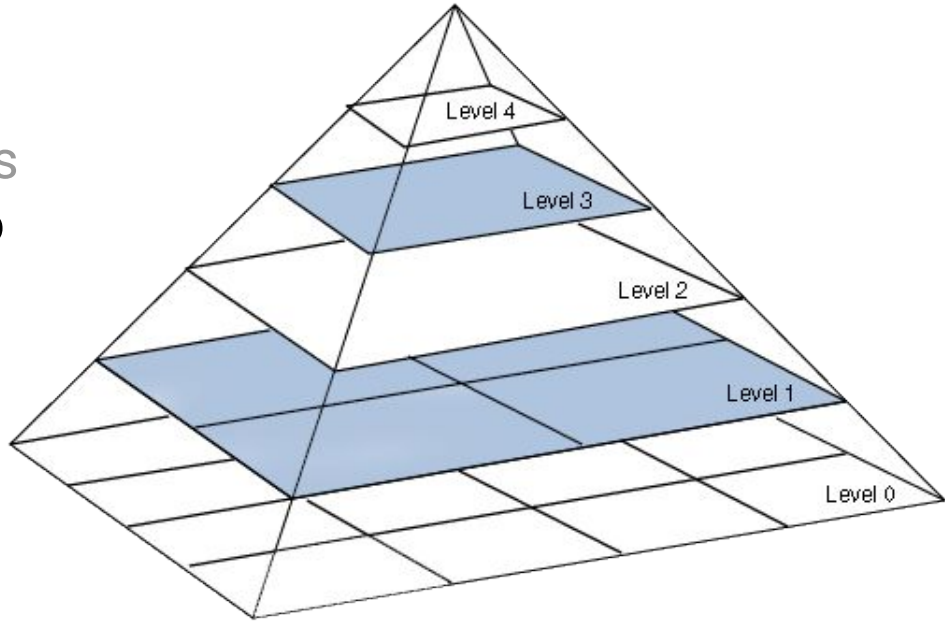
Ingesting EE Images

1. Upload input scene to Google
2. Leave scene in original projection and resolution
3. Break the image into 256 x 256 tiles
4. Downsample the tiles repeatedly to produce a...



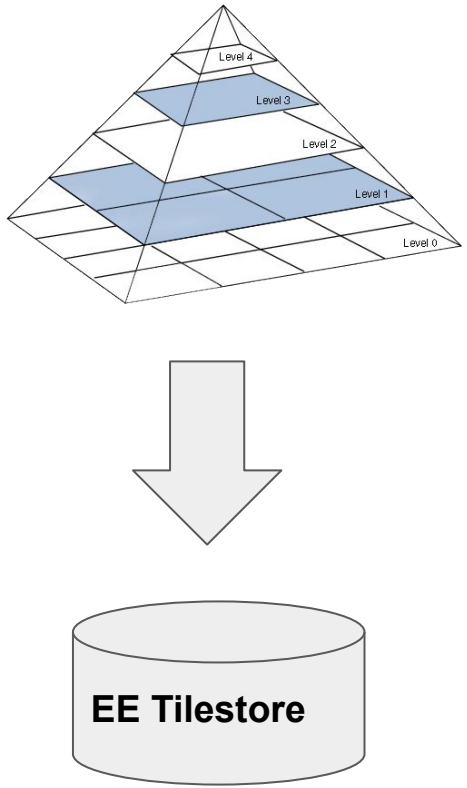
Ingesting EE Images

1. Upload input scene to Google
2. Leave scene in original projection and resolution
3. Break the image into 256 x 256 tiles
4. Downsample the tiles repeatedly to produce a image pyramid



Ingesting EE Images

1. Upload input scene to Google
2. Leave scene in original projection and resolution
3. Break the image into 256 x 256 tiles
4. Downsample the tiles repeatedly to produce a image pyramid
5. Store the full image pyramid in the tile datastore.



Accessing and Analyzing EE Images

Ok so now there are millions of image pyramids in EE, with a huge variety of map projections at any possible resolution.

How can I perform meaningful analysis if there are no guarantees about a consistent projection or pixel size?

Answer: All retrieved image tiles are processed in the output projection and scale

"**Output**" means either :

- a function parameter (e.g. crs),
- the Map in the Code Editor,
- or with a reproject() call

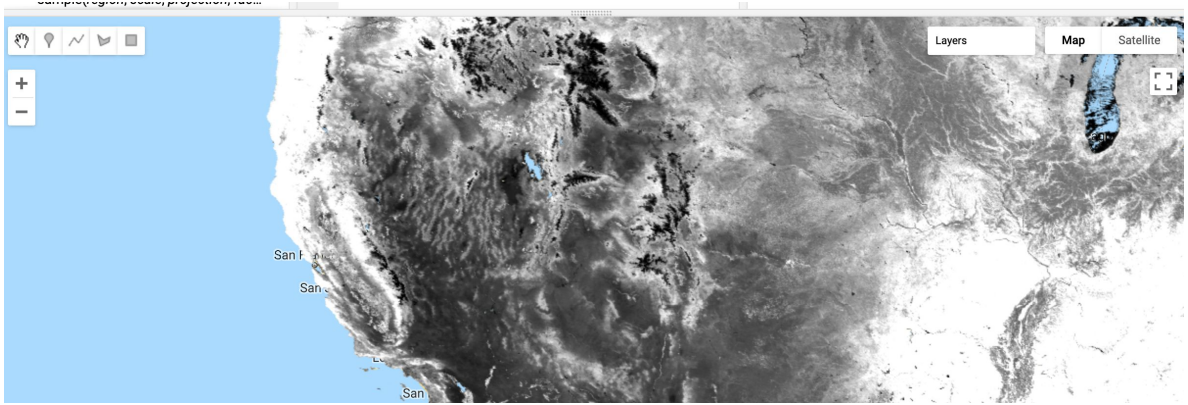
Earth Engine almost always handles the reprojecting and scaling. So **you (usually) don't have to worry about it.**

Accessing and Analyzing EE Images

What happens when we add an image to the map?

```
// The input image has a SR-ORG:6974 (sinusoidal) projection.
var image = ee.Image('MODIS/006/MOD13A1/2014_05_09').select(0);

// Normalize the image and add it to the map.
var rescaled = image.unitScale(-2000, 10000);
var visParams = {min: 0.15, max: 0.7};
Map.addLayer(rescaled, visParams, 'Rescaled');
```



Input: Modis (Sinusoidal Projection)

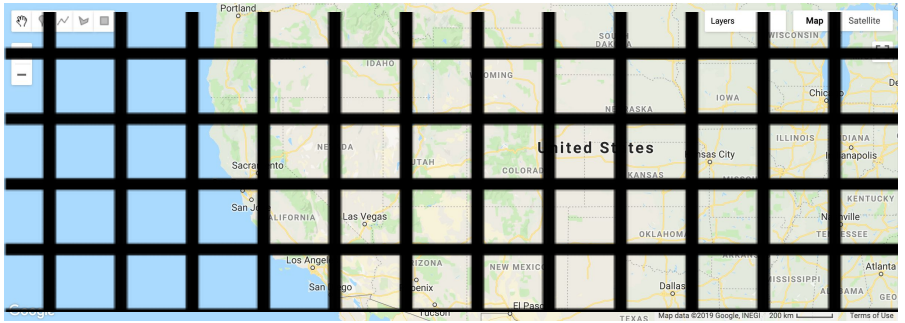
Output: Code Editor Map (WGS84 projection)

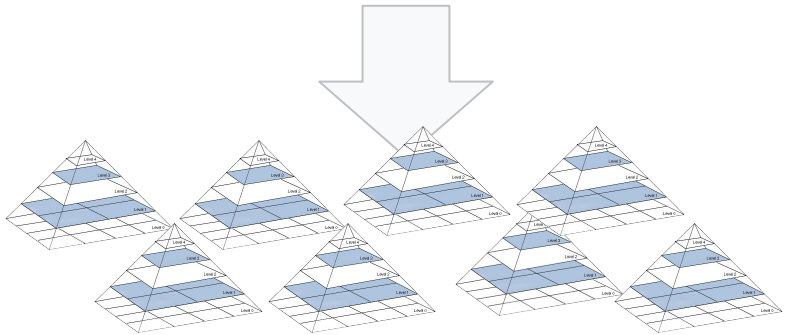
All computations will happen in the **Output** projection

Accessing and Analyzing EE Images

What happens when we add the 'rescaled' image to the map?

1. Retrieve the image pyramids for the tiles that are currently in view on the Map

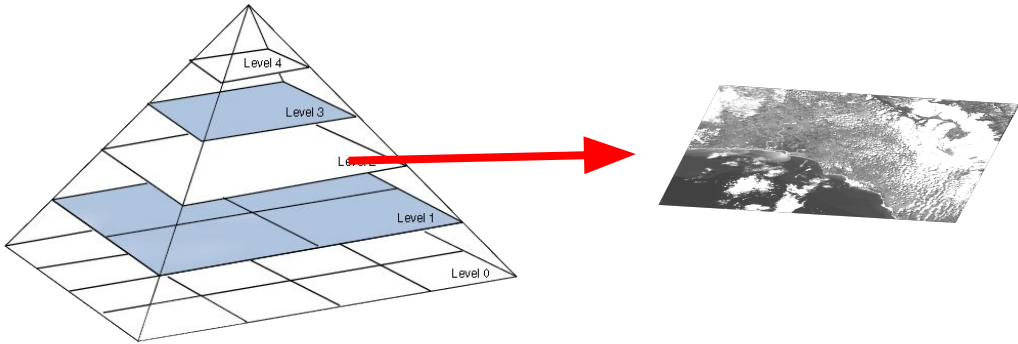




Accessing and Analyzing EE Images

What happens when we add the 'rescaled' image to the map?

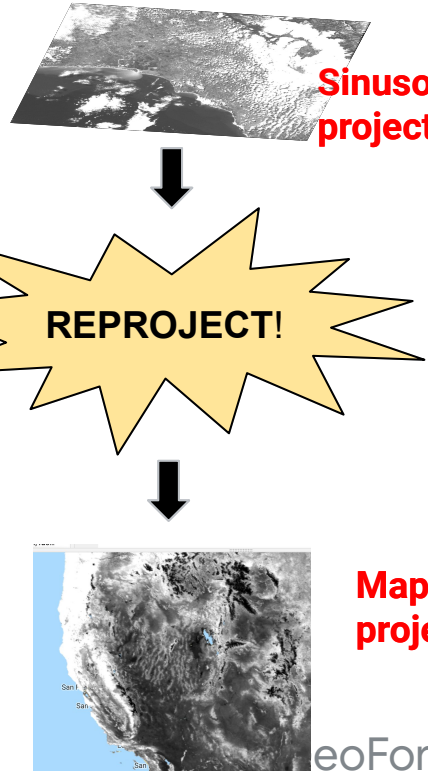
1. Retrieve the image pyramids for the tiles that are currently in view on the Map
2. Choose the pyramid level closest to Map's zoom level. Downsample pixels from there



Accessing and Analyzing EE Images

What happens when we add the 'rescaled' image to the map?

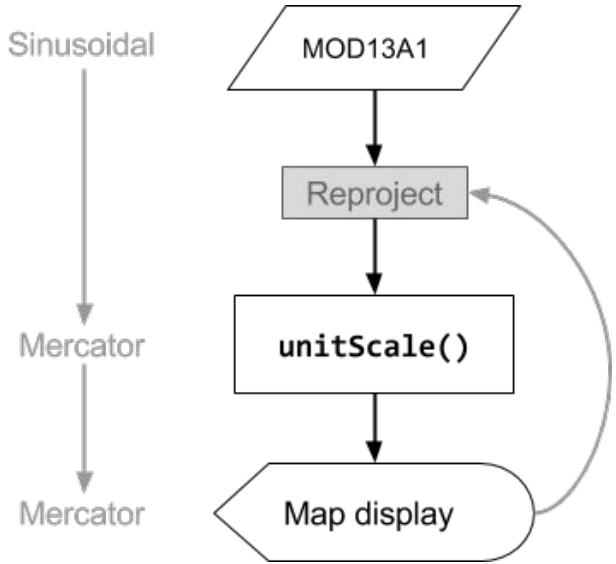
1. Find the image pyramids for the tiles that are currently in view on the Map
2. Choose the pyramid level closest to Map's zoom level. Downsample pixels from there
3. Reproject each raster into the output projection



Accessing and Analyzing EE Images

What happens when we add the 'rescaled' image to the map?

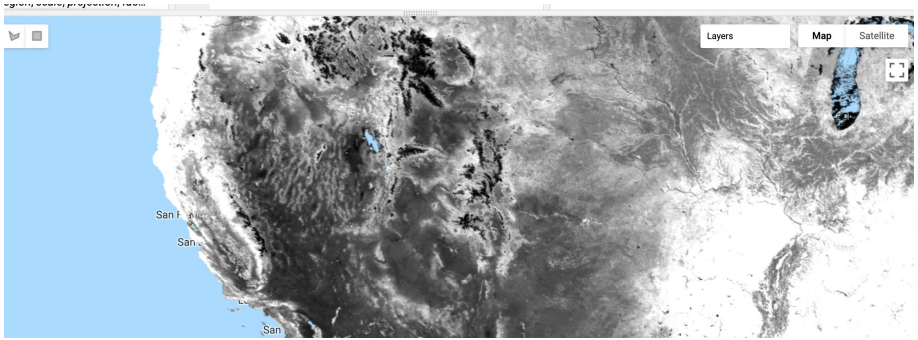
1. Find the image pyramids for the tiles that are currently in view on the Map
2. Choose the pyramid level closest to Map's zoom level. Downsample pixels from there
3. Reproject each raster into the output projection
4. Perform computations on the newly reprojected pixel tiles



Accessing and Analyzing EE Images

What happens when we add the 'rescaled' image to the map?

1. Find the image pyramids for the tiles that are currently in view on the Map
2. Choose the pyramid level closest to Map's zoom level. Downsample pixels from there
3. Reproject each raster into the output projection
4. Perform computations on the newly reprojected pixel tiles
5. Send results to Code Editor to be displayed on screen



Thus concludes our journey of into the heart of Earth Engine

… now back to what we can do with Images and Features

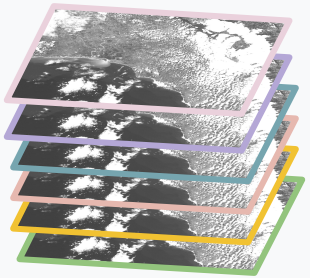
To Review: EE Images are basically "**containers**"

```
ee.Image('CGIAR/SRTM90_V4')
```

Properties

```
{
  date: 12345,
  footprint: [12…],
  name: 'my_image'
}
```

Bands



Inspector | Console | Tasks

```
Image CGIAR/SRTM90_V4 (1 band)
  type: Image
  id: CGIAR/SRTM90_V4
  version: 1550150077793073
  bands: List (1 element)
    0: "elevation", signed int16, E
  properties: Object (19 properties
    date_range: [950227200000,95117
    period: 0
    product_tags: List (5 elements)
    provider: NASA/CGIAR
    provider_url: http://srtm.csi.c
    sample: https://mw1.google.com/
```

Pixels in an Image can be **masked**

Masking pixels in an image makes those pixels **transparent.**

Masked pixels are not included in computations or visualizations.

updateMask(mask)

Updates an image's mask at all positions where the existing mask is not zero. The output image retains the metadata and footprint of the input image.

selfMask()

Updates an image's mask at all positions where the existing mask is not zero using the value of the image as the new mask value. The output image retains the metadata and footprint of the input image.

unmask(*value*, *sameFootprint*)

Replaces mask and value of the input image with the mask and value of another image at all positions where the input mask is zero.

Demo - The default mask

```
// Select RGB bands from landsat image
var image = ee.Image('LANDSAT/LC08/C01/T1_TOA/LC08_042032_20130322')
                  .select(['B4', 'B3', 'B2'])

// Style and add to the map
var viz = {min: 0.0, max: 0.4};
Map.addLayer(image, viz)
```

What is the default mask?

```
Map.addLayer(image.mask())
```

Unmasked pixels have value of 1. Masked have **no value.**

Demo - Updating the Mask

```
// Mask out any pixels greater than .2
image = image.updateMask(image.lt(.2))
Map.addLayer(image)
```

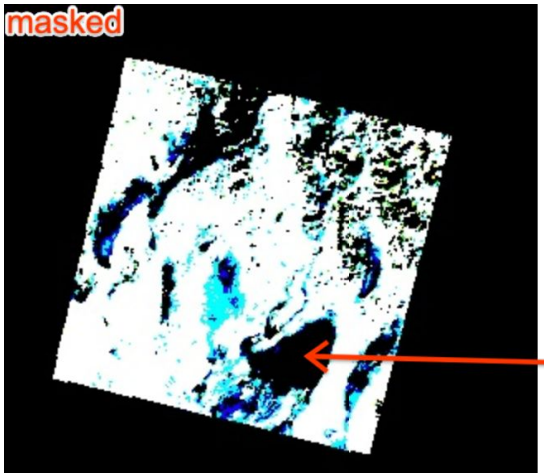
Original image

With new mask applied

Demo- What does the new mask look like?

```
// Mask out any pixels greater than .2
image = image.updateMask(image.lt(.2))
Map.addLayer(image.mask())
```

Original mask

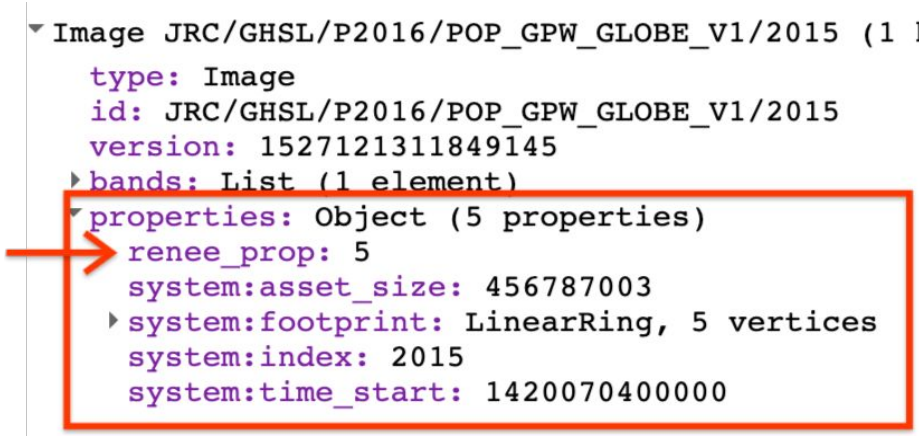


Updated mask

Image Properties: A dictionary or "hashmap" of key value pairs.

```
var ghslPop = ee.Image('JRC/GHSL/P2016/POP_GPW_GLOBE_V1/2015');

ghslPop = ghslPop.set('renee_prop', 5);
print(ghslPop)
```

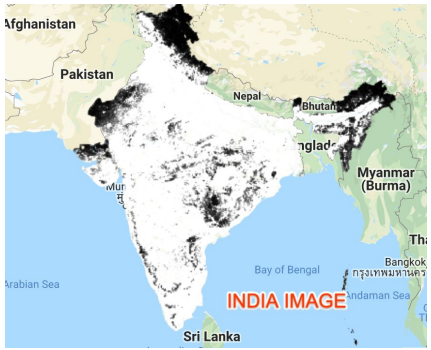


Some image properties are built-in

Example: **system:footprint** determines image geometry
Any pixels outside of the "footprint" are masked.

```
var india = ee.Image('WorldPop/GP/100m/pop/IND_2015')

var footprint = ee.Geometry(india.get('system:footprint'))
var geometry = india.geometry()
var boundingBox = india.geometry().bounds()
```



Some image properties are built-in

Example: **system:time** determines image Date

```
var india = ee.Image('WorldPop/GP/100m/pop/IND_2015')
var timeStamp = india.get('system:time_start')
var date1 = ee.Date(timeStamp)
var date2 = india.date()

print('timestamp', timeStamp);
print('From timestamp', date1);
print('From date', date2);
```

```
timestamp
1420070400000

From timestamp
Date (2015-01-01 00:00:00)

From date
Date (2015-01-01 00:00:00)
```

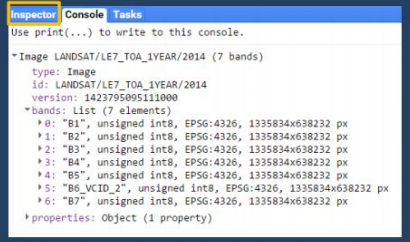
Image Bands

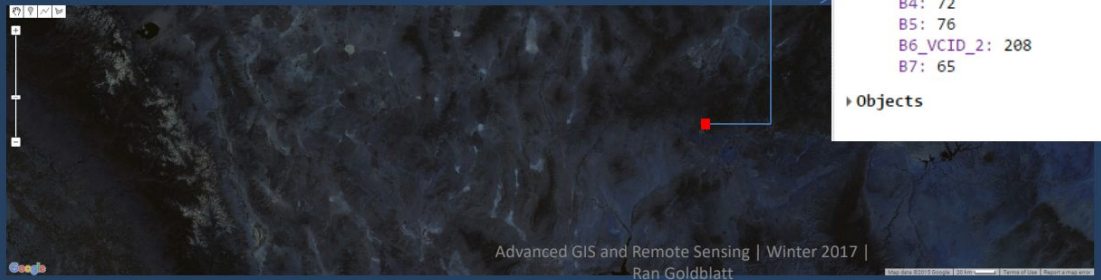
- A 2D grid of **Pixels.**
- Pixels in a single band must share the same:
  - Pixel **data type** (boolean, int, float, array)
  - **Resolution** (ex: 30m pixels)
  - **Projection** (ex: WGS84, EPSG:4326)
- Images can contain **many bands**.
- Each band is stored as its own image pyramid which means they lend themselves well to **parallel computation**.

Visualizing Bands on the Map

Many images in Earth Engine have one or more bands.
For example, images from the Landsat 7 satellite include:

Band 1 - blue
Band 2 - green
Band 3 - red
Band 4 - Near Infrared
Band 5 - Short-wave Infrared
Band 6 - Thermal Infrared
Band 7 - Short-wave Infrared





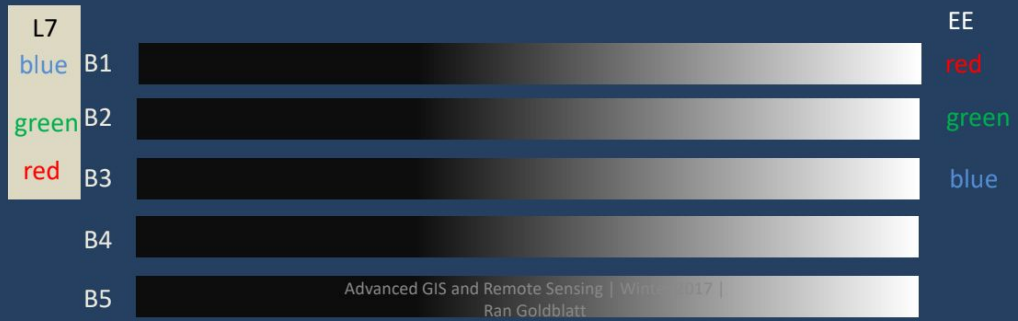
[Source link]

Visualizing Bands on the Map

How does EE map the values in the image bands to colors on the map?

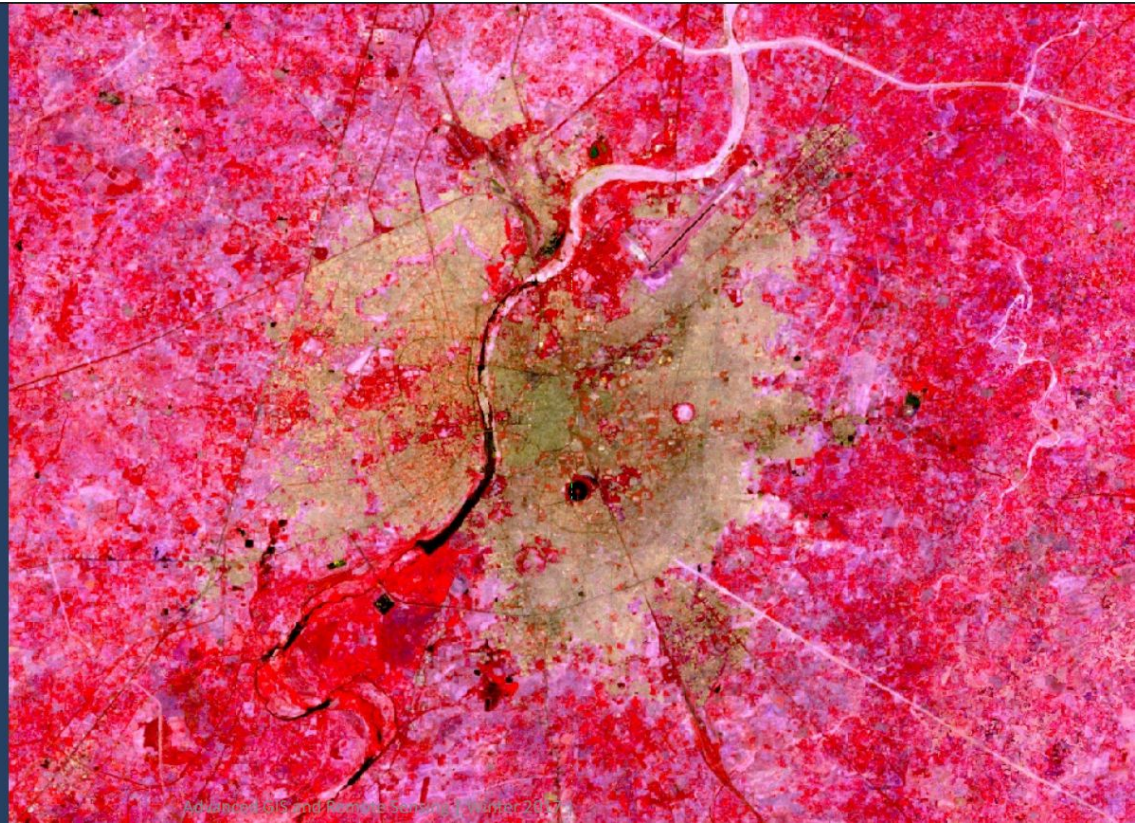
One band: value 0 is rendered as black and value 255 as white, with a linear gray gradient in between.

More than one band:
EE takes the first three bands and maps the first to red, the second to green, and the third to blue. The value of the pixel determines the intensity of the color.

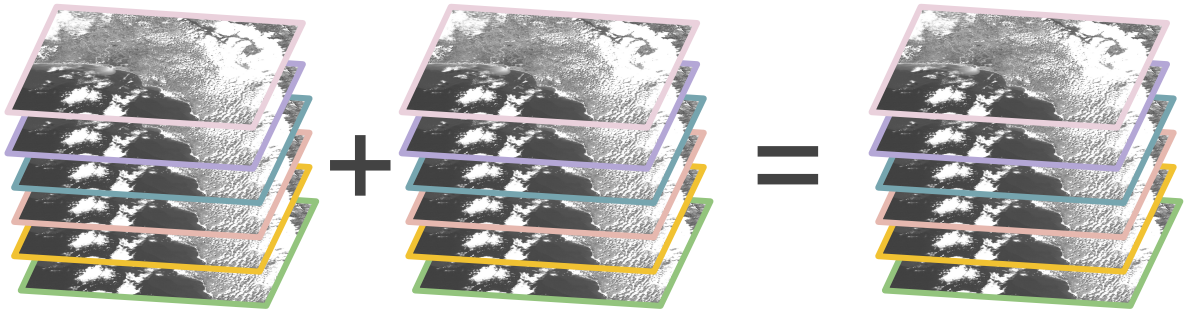


Example of a non-RGB visualization

Landsat 7:
4,3,7
(NIR, RED, SWIR)



Band Operations: Math



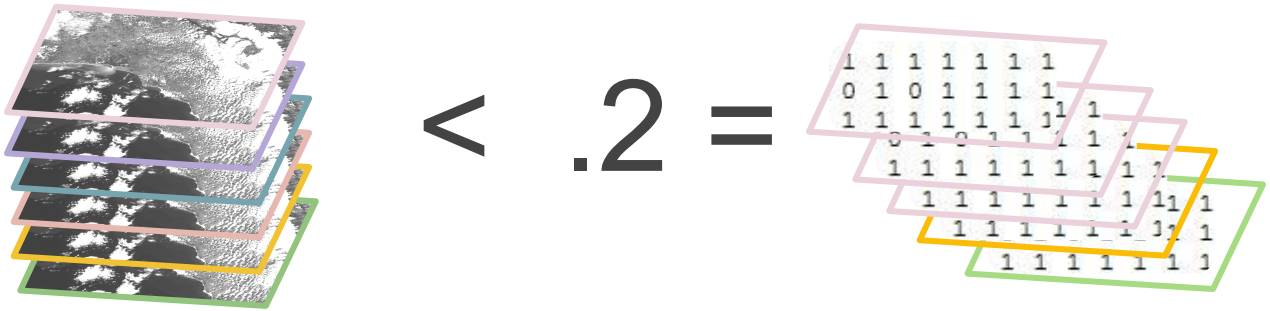
Many basic operations available:

Add, Subtract, Multiply, Divide, Power

Plus some custom "short cut" operations:

Image.normalizedDifference()

Band Operations: Relational, Conditional, Boolean



Comparisons:

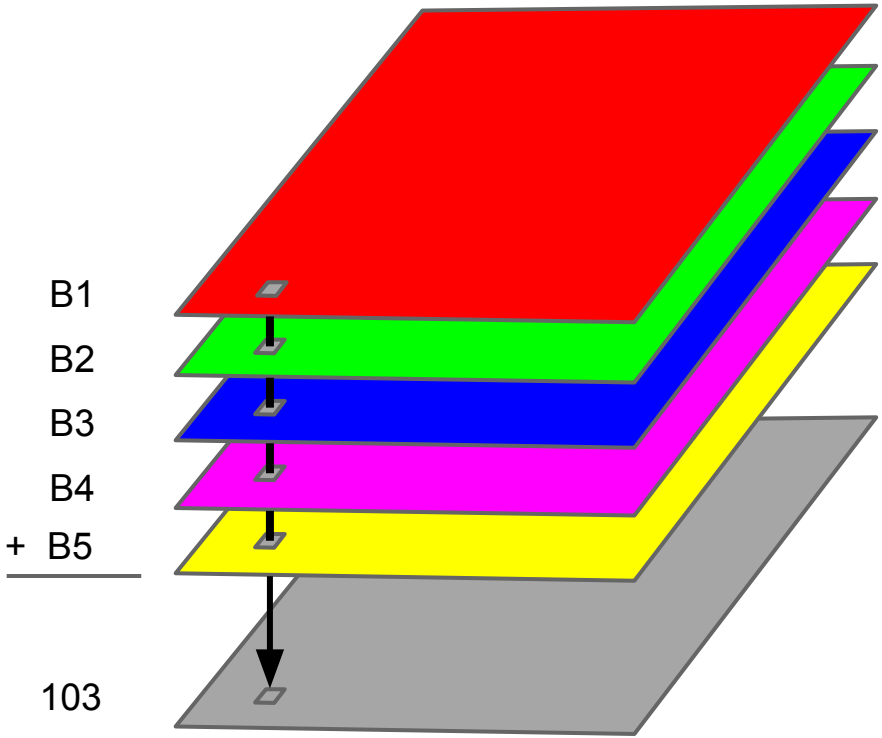
- gt(), lt(), eq(), neq()

- or(), and(), not()

Image Operations: Advanced Image Processing

- **image.convolve()** - for linear convolutions/filtering of pixels
- **image.focal_max()** - morphological operations (also min, median)
- image.gradient()
- Spectral Transformations
  - image.normalizedDifference()
  - image.unmix()
  - image.rgbToHsv()
  - image.hsvToRgb()
- Spatial texture methods

Image Reductions: Reduce bands



Reducers in Earth Engine

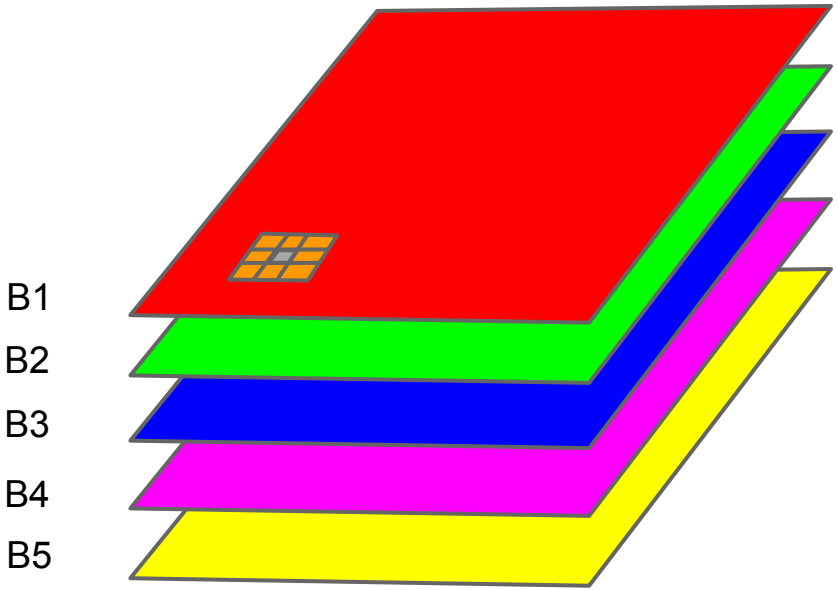
8 ways to reduce

Image.reduce
Image.reduceNeighborhood
Image.reduceRegion
Image.reduceRegions
Image.reduceToVectors
ImageCollection.reduce
FeatureCollection.reduceColumns
FeatureCollection.ReduceToImage

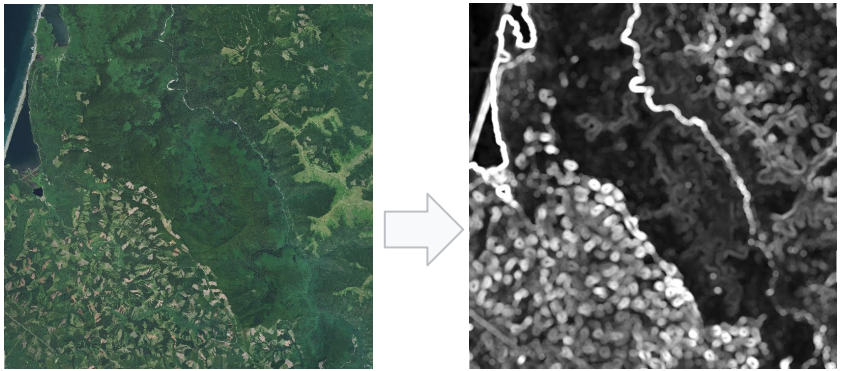
40+ reducers

Reducer.allNonZero
Reducer.and
Reducer.anyNonZero
Reducer.count
Reducer.countEvery
Reducer.histogram
Reducer.intervalMean
Reducer.linearFit
Reducer.linearRegression
Reducer.max
Reducer.mean
Reducer.median
Reducer.min
Reducer.minMax
Reducer.mode
Reducer.or
Reducer.percentile
Reducer.product
Reducer.sampleStdDev
Reducer.sampleVariance
Reducer.stdDev
Reducer.sum
Reducer.toCollection
Reducer.toList
Reducer.variance

Reduce Neighborhoods (Kernels)




```
// Compute standard deviation (SD) as texture of the NDVI.
var texture = naipNDVI.reduceNeighborhood({
  reducer: ee.Reducer.stdDev(),
  kernel: ee.Kernel.circle(7),
});
```



https://developers.google.com/earth-engine/reducers_reduce_neighborhood

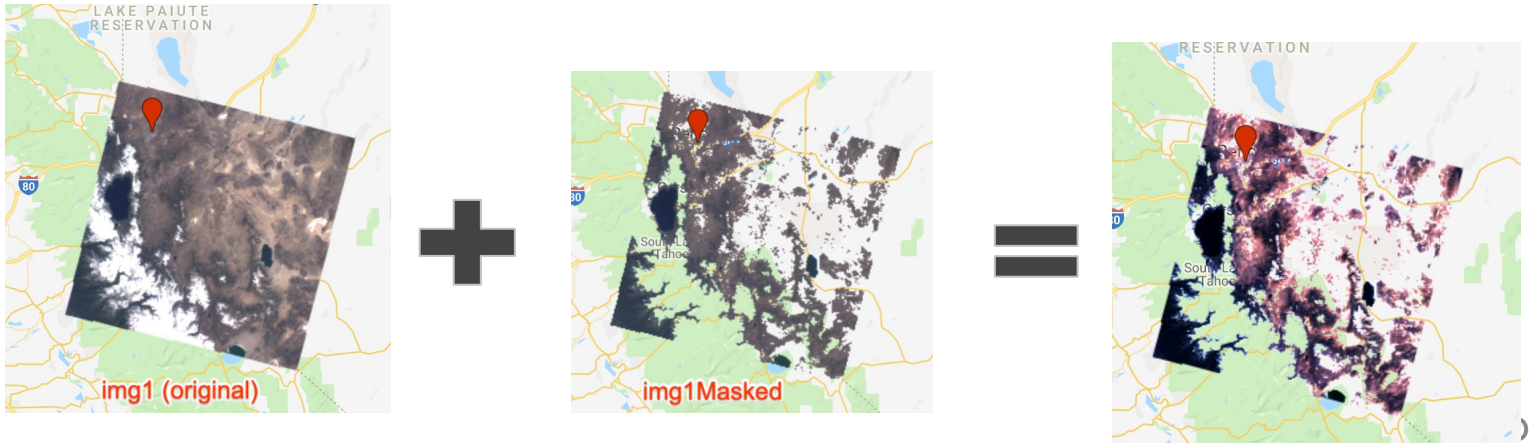
~~ INTERMISSION ~~

EE Image Trivia

```
var newImage = image1.add(image2)
```

Q: What if the pixels in one band are masked, but are not masked in the other?

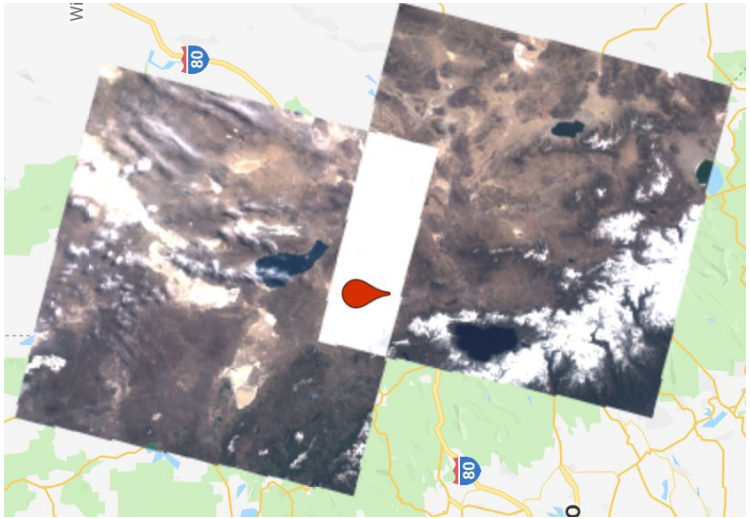
A: The masked area stays masked. You get the **intersection** of the unmasked area.



EE Image Trivia

```
var newImage = image1.add(image2)
```

Q: What happens if image1 and image2 have different footprints (geometries)?
A: You get the **intersection** of the two images



Important Note

```
var newImage = image1.add(image2)

var newImage = ee.ImageCollection([image1, image2]).sum()
```

These handle masking and footprints differently!

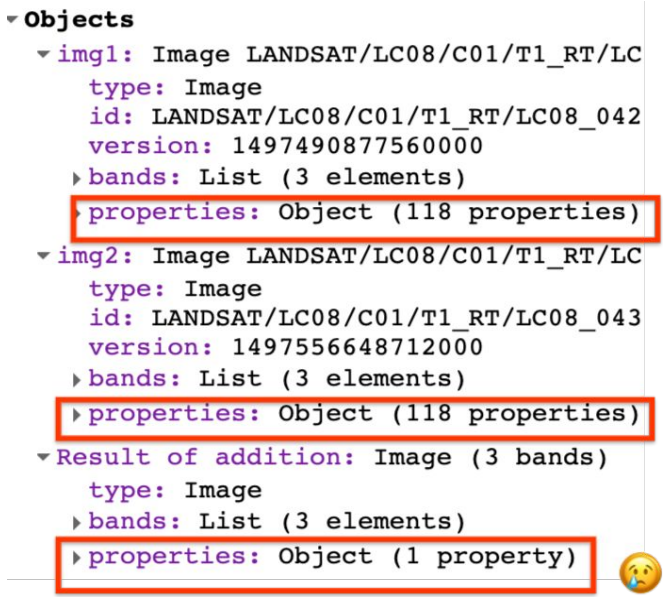
- **Adding** two images -> **intersection** of unmasked pixels
- **Reducing an ImageCollection** of those two images -> **union** of unmaked pixels

EE Image Trivia

```
var newImage = image1.add(image2)
```

Q: **image1** has a different "date" property than **image2**. What happens?

A: All properties get dropped! (Use **copyProperties()**)



EE Image Trivia

```
var newImage = image1.add(image2)
```

Q: What if the images don't have the same number of bands?
A: Error!

```
img1 added to other band order: Layer error:
Image.add: Images must contain the same number of
bands or only 1 band. Got 3 and 2.
```

EE Image Trivia

```
var newImage = image1.add(image2)
```

Q: What if the bands I want to match are in different orders? ([B1, B2, B3] vs [B3, B2, B1]?

A: B1 and B3 will get added together which is probably not what you want.

Image Brain Teasers

Q: Can an image ever be too big?

A: Guideline is to not have more than 2K bytes per pixel across all bands.

A: If you have too many bands

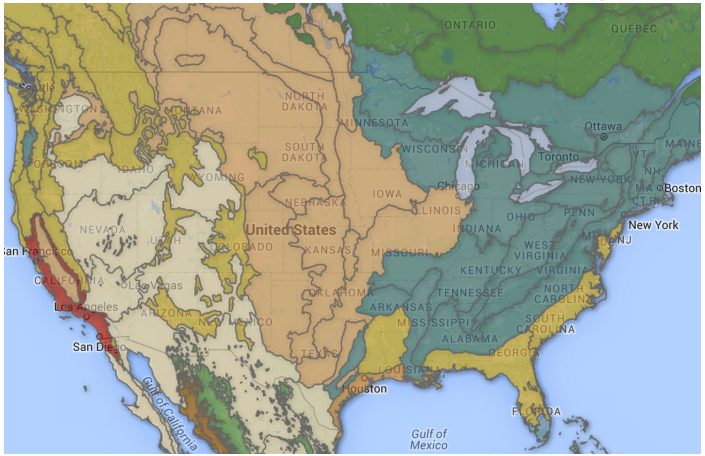
A: If you have humongous array-valued pixels

A. Memory intensive pixels may still cause problems even if they don't hit the limit. Consider retile()

Earth Engine Features

Earth Engine Features

TNC Ecoregions



Another "bag" containing:

- A **Geometry-** Line / Point / Polygon
- List of **Properties-** Just like images
- Basically a **row** in a **table** with a geometry (.geo) column

id	.geo	city	country	population	year
"my_feature"	Point(1,2)	Reno	USA	300,000	2018

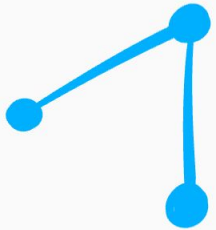
Earth Engine Geometries

(Huge thanks to Sai Cheems for the following slides)

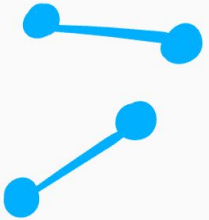
Primitives

→ point

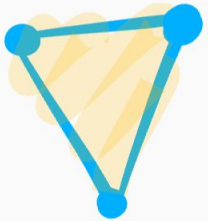
→ multipoint



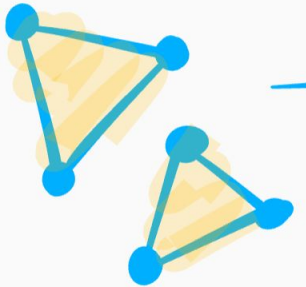
→ line



→ multiline

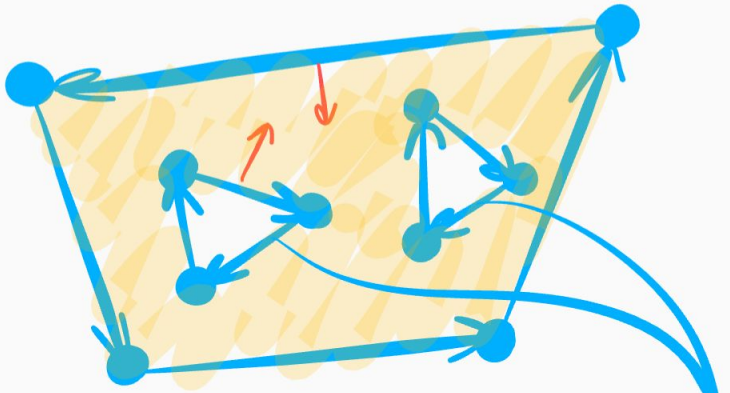


→ polygon



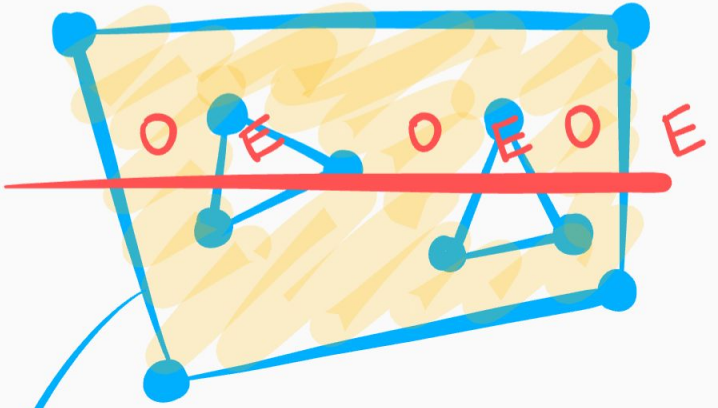
→ multipolygon

Polygons



Inside on the left

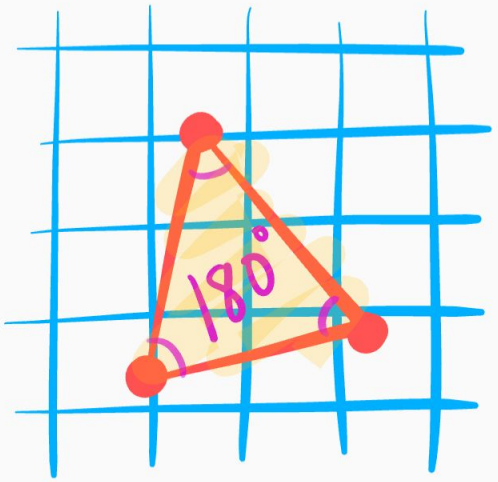
holes

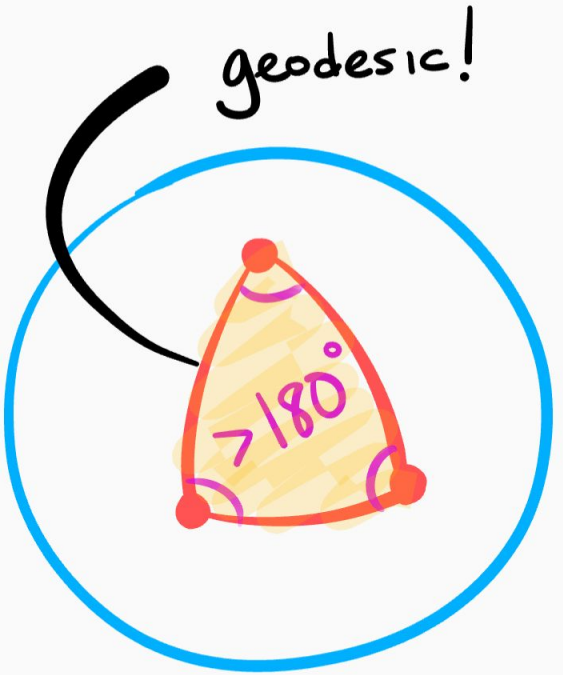


Even odd

shell

2D/Sphere

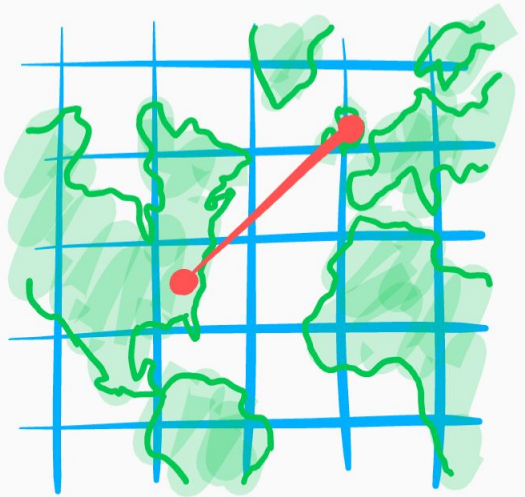




(X, Y) coordinates on a Cartesian plane

(X, Y, Z) coordinates on a sphere

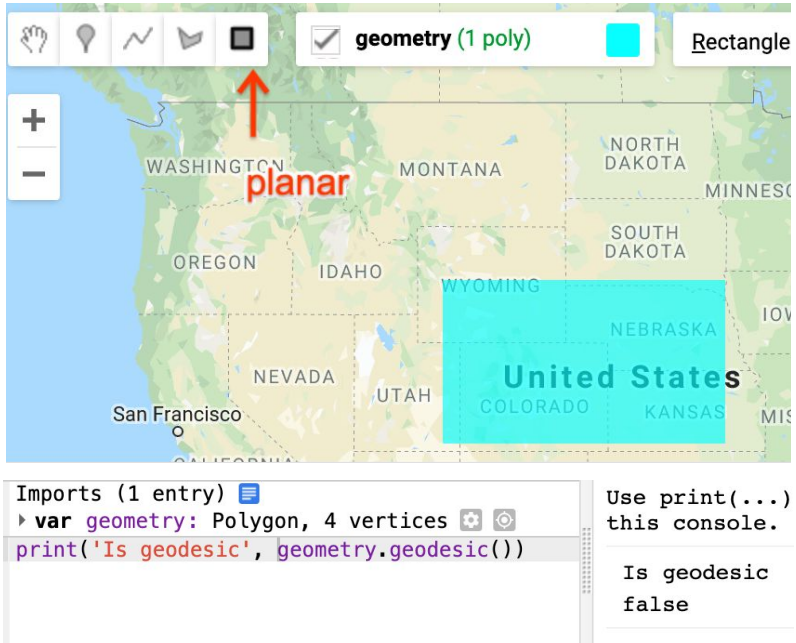
Geodesics



→ A geodesic is the shortest path between 2 points on a surface

Geometries in the Code Editor can be Planar or Geodesic

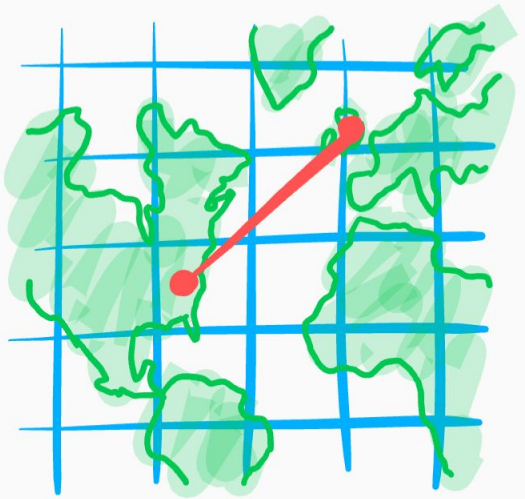
Points are transformed to spherical coordinates for calculations



Tessellation

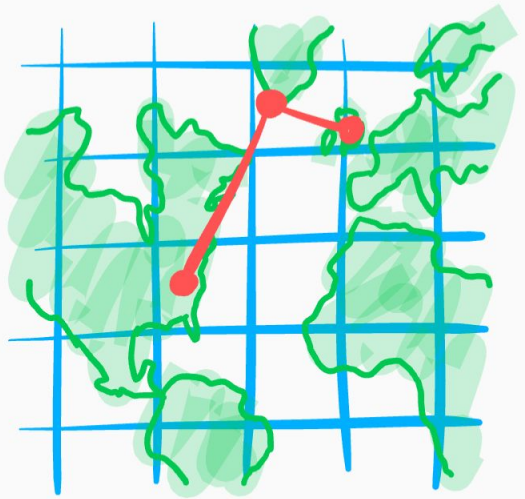
Station

Tessellation



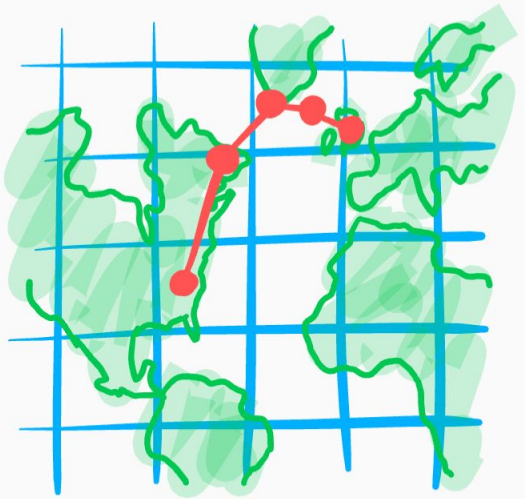
→ Just transforming points is not enough to get an accurate representation of a geodesic in 2D

Tessellation



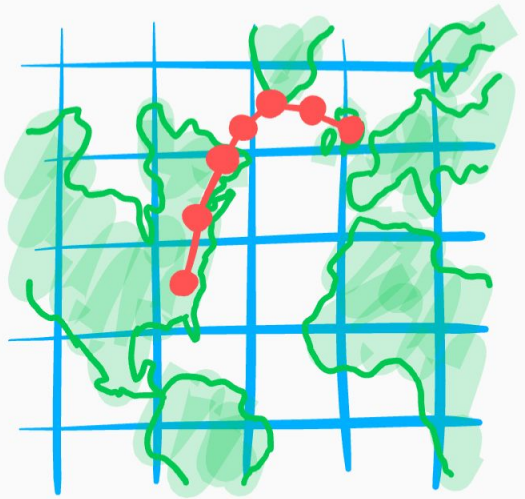
→ Our strategy is to split recursively until the error of each resulting edge is below a user-specified margin

Tessellation



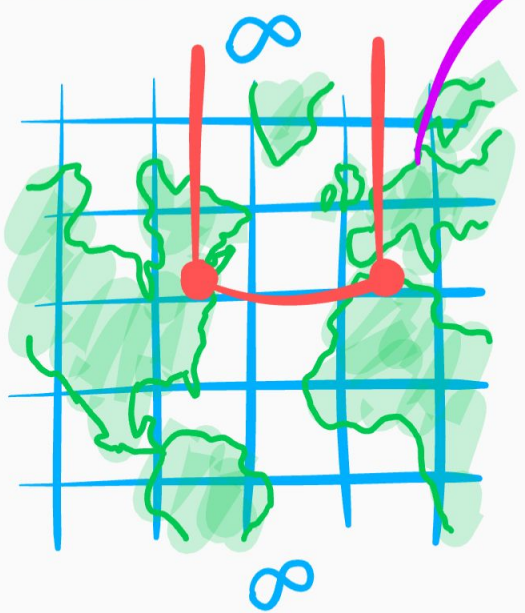
→ Our strategy is to split recursively until the error of each resulting edge is below a user-specified margin

Tessellation

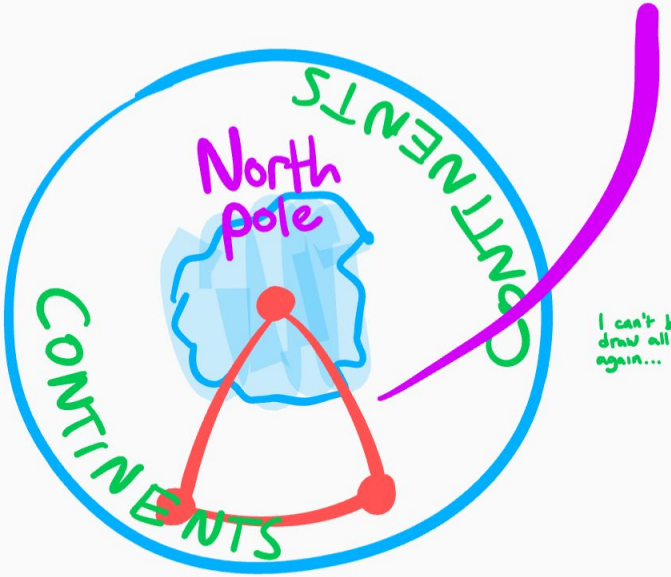


→ Our strategy is to split recursively until the error of each resulting edge is below a user-specified margin

Tessellation



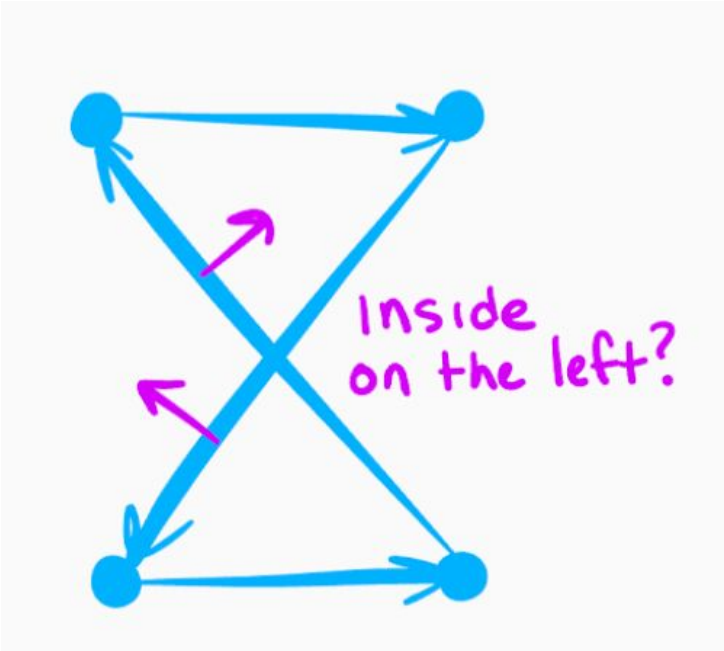
Tessellating this line is a deathtrap

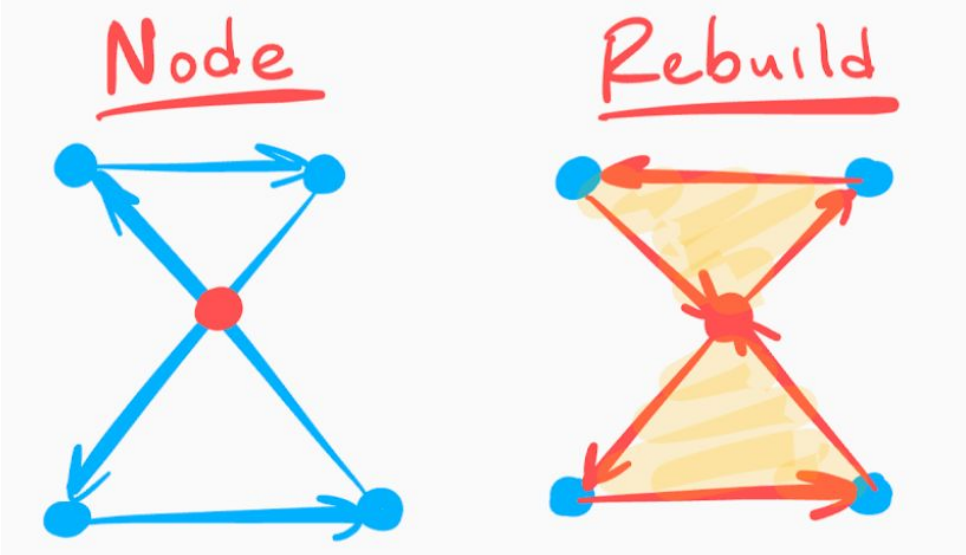


I can't be bothered to draw all those continents again...

→ Maps Mercator is a cylindrical projection which we truncate at ~85°. That's because the poles project towards infinity

Geometries that cause problems

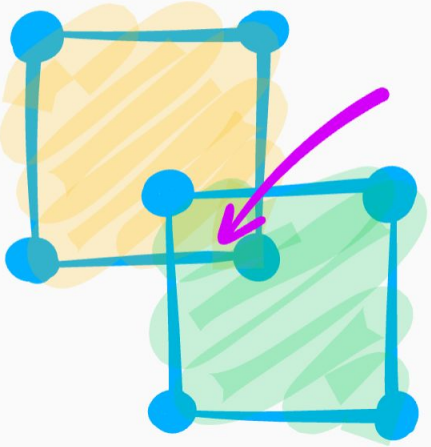




EE's solution is to remake the geometry.
This might not be what you want.

Operations

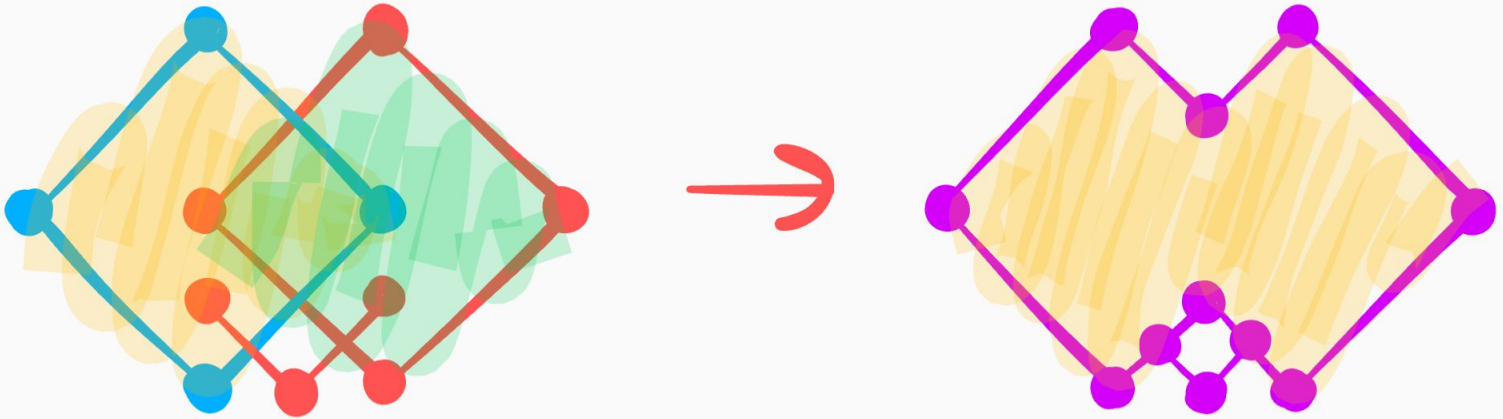
¯\(ツ)/¯



- In Earth Engine, the area of the geometry on the right is the yellow + green areas
- Many other tools dissolve first (the area is just the yellow + green areas − the overlap
- This concept holds in other operations. We don't dissolve for you

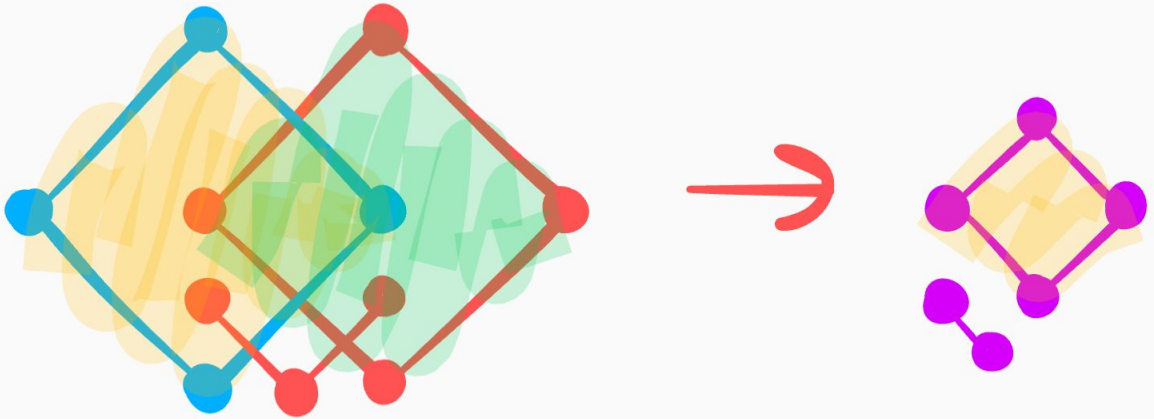
Union / Dissolve

blue.union(red)



Intersection

blue.intersects(red)



Feature Operations

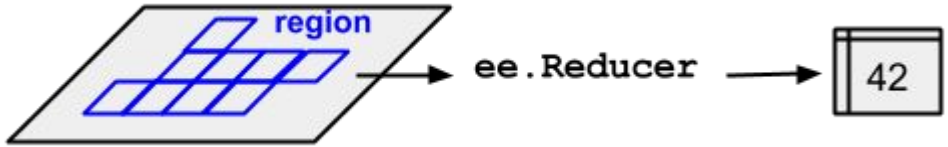
Most Feature operations are just geometric operations

Feature-related Operations:

- propertyNames, update properties
- **reduceRegion**()

Reduce Region

To get statistics of pixel values in a region of an `ee.Image`, use `image.reduceRegion()`



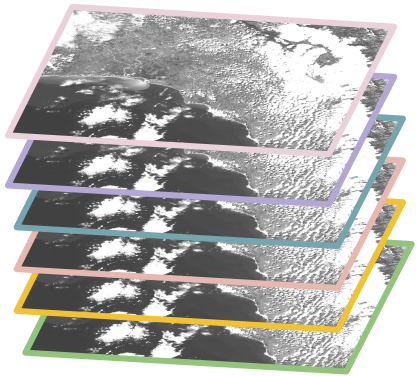

```
// Reduce the region. The region parameter is the Feature geometry.
var meanDictionary = image.reduceRegion({
  reducer: ee.Reducer.mean(),
  geometry: region.geometry(),
  scale: 30,
  maxPixels: 1e9
});
```

Compute everything at 30m per pixel resolution

Override the max allowed pixels to compute

Reduce Region

The output of **`reduceRegion`** is a **dictionary** of the results for **each band.**



```
image.reduceRegion(
  ee.Reducer.sum()
)
```


```
{
  "B1": 80256763.12156858,
  "B10": 208590789.23921564,
  "B11": 190384450.3647058,
  "B2": 76990288.55686265,
  "B3": 74459424.74117647,
  "B4": 77791588.67843133,
  "B5": 100857336.23137248,
  "B6": 100739323.8588235,
  "B7": 87230669.56078431,
  "B8": 75159903.6784314,
  "B9": 43466944.803921536,
  "BQA": 23761121.505882356
}
```

To Recap

What are Images and Features?

Just "bags" to hold either:
- Properties and raster bands (Images)
- Properties and a geometry (Feature)

Where to go next?

Bags of Bags:
- "EE Collections and map()"
- "EE Reducers"
- "EE Tables & Joins"

Thanks! Questions?

Next Steps

Check out the Code Lab
bit.ly/2mjYU7j