Raster Reprojection, No Resampling Required

(or, how to hack the geotransform for quality and speed)

Tuesday, Oct. 24, 2023, 11:30am - noon
Holiday Ballroom 1
FOSS4GNA 2023 Baltimore

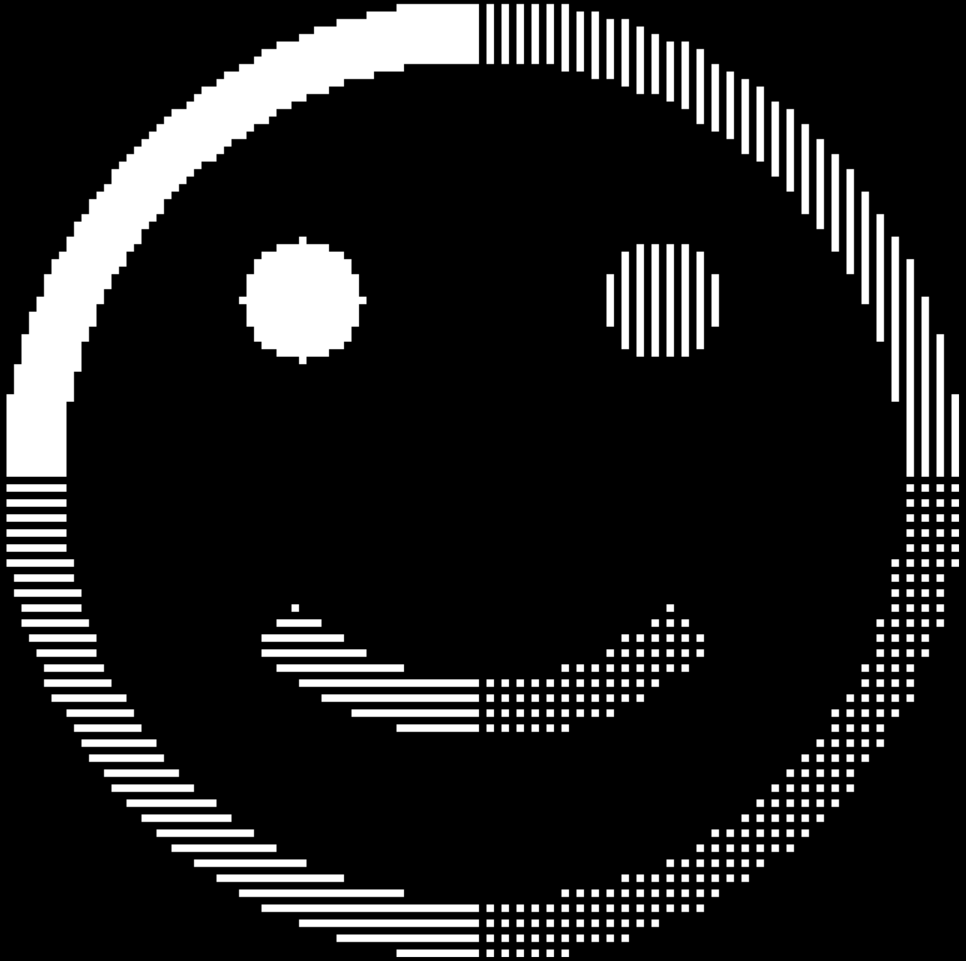
Dan Mahr
Lead Software Engineer @ DroneDeploy

Sample image

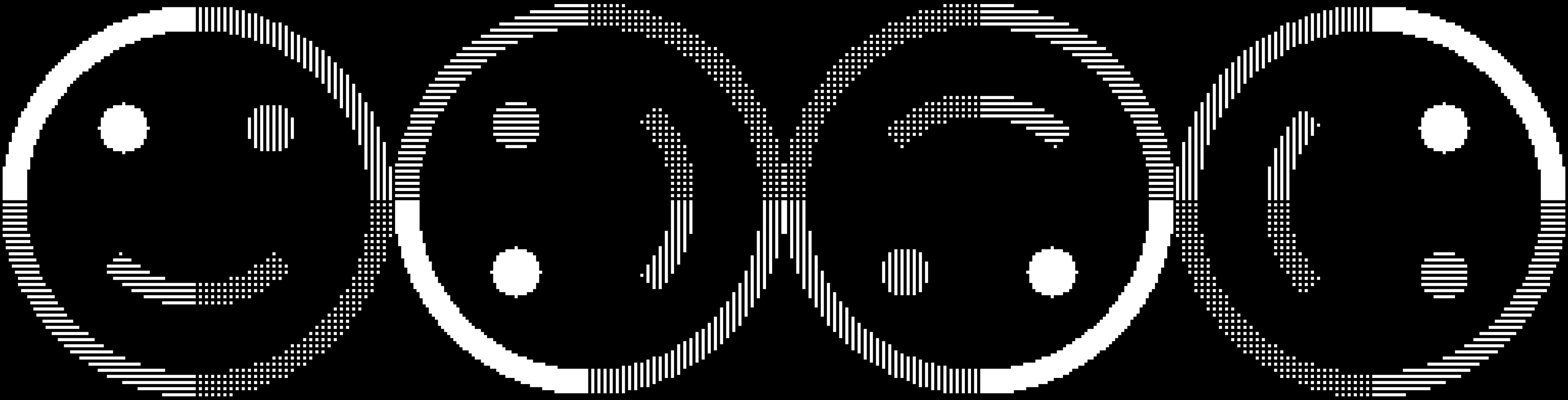
128 x 128 pixels.

Stripes are intentional.

All sample images in this presentation have been upscaled 8x to so that Google Slides doesn't apply its own resampling.

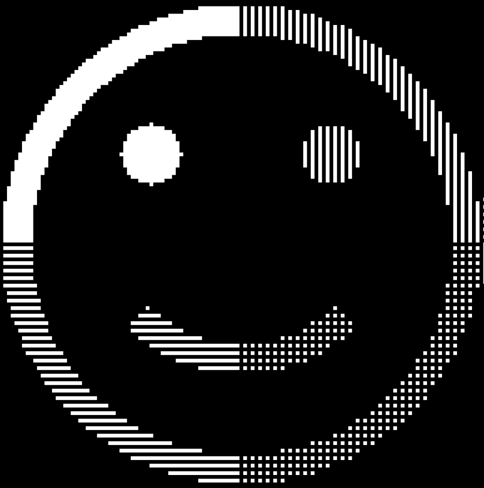


Lossless operations: rotation in 90° increments

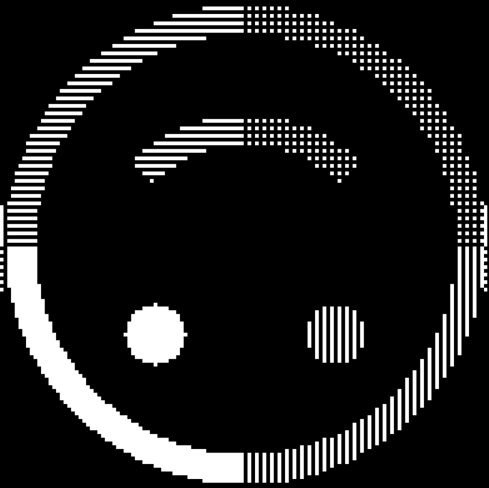


Original image **Rotate 90° CCW** **Rotate 180°** **Rotate 90° CW**

Lossless operations: reflect

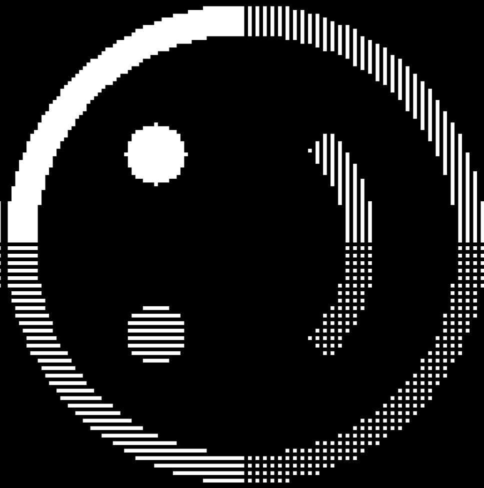


Original image



Vertical flip

Horizontal flip



Transpose

Lossless operations: enlarge by integer

Original image

2x upsampled

Every pixel in original image becomes 4 pixels.

Lossy operation: rescale by non-integer

Original image

4/3x upsample

Nearest neighbor resampling

Lossy operation: rescale by non-integer

Original image

4/3x upsample

Bilinear resampling

Lossy operation: rescale by non-integer

Original image

4/3x upsample

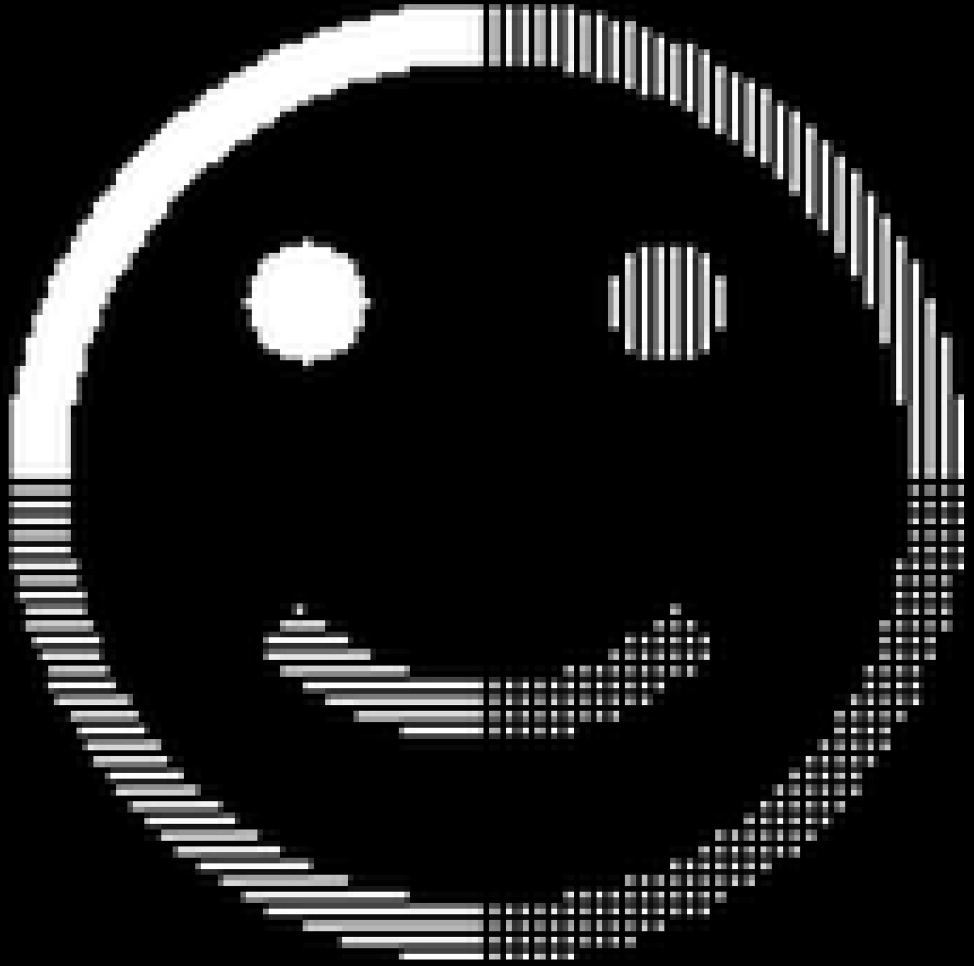
Bicubic resampling

Lossy operation: rescale by non-integer

Original image

4/3x upsample

Lanczos resampling



Lossy operation: rotate, not axis-aligned

Original image

45° rotation

Nearest neighbor resampling

Lossy operation: rotate, not axis-aligned

Original image

45° rotation

Bilinear resampling

Georeferencing

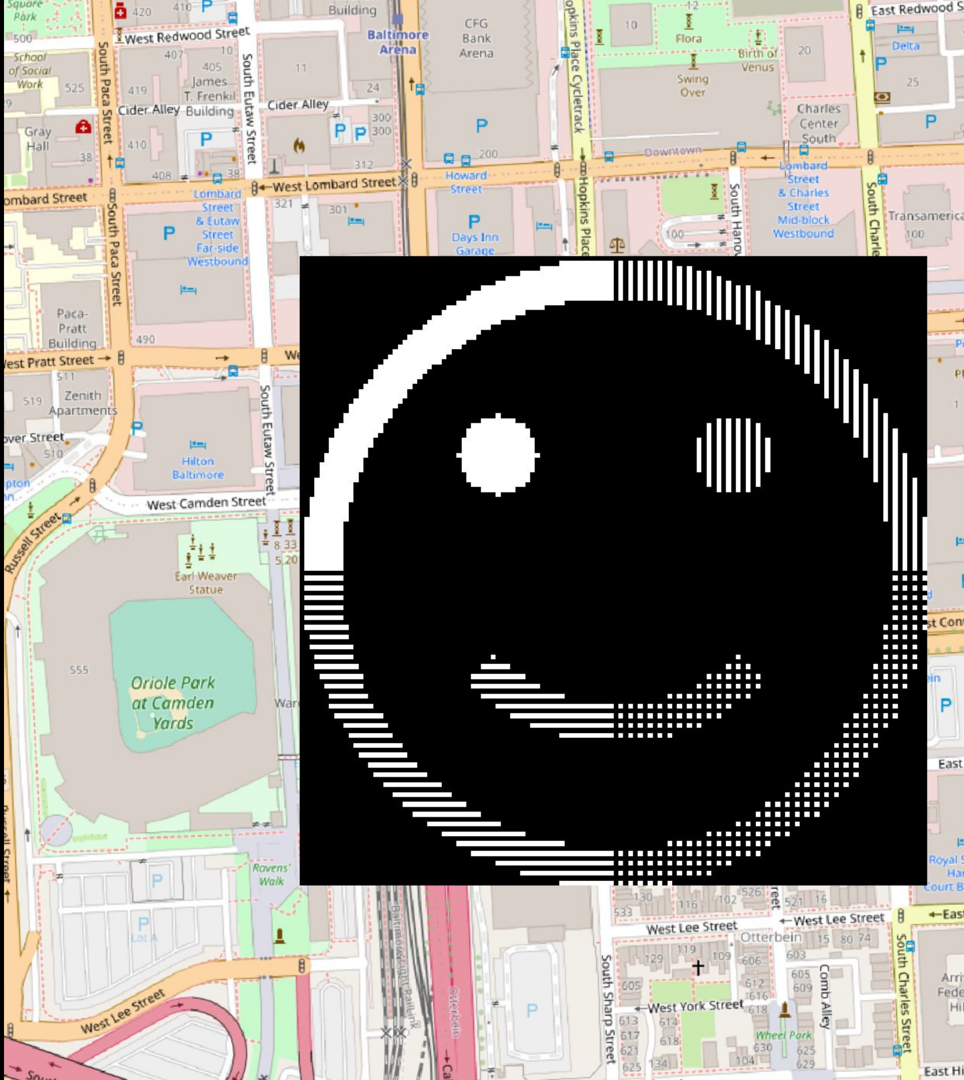
```
$ gdal_edit.py
-a_srs epsg:26985
```

NAD83 / Maryland projected CRS in meters.

```
-a_ullr
432750 179936
433262 179424
```

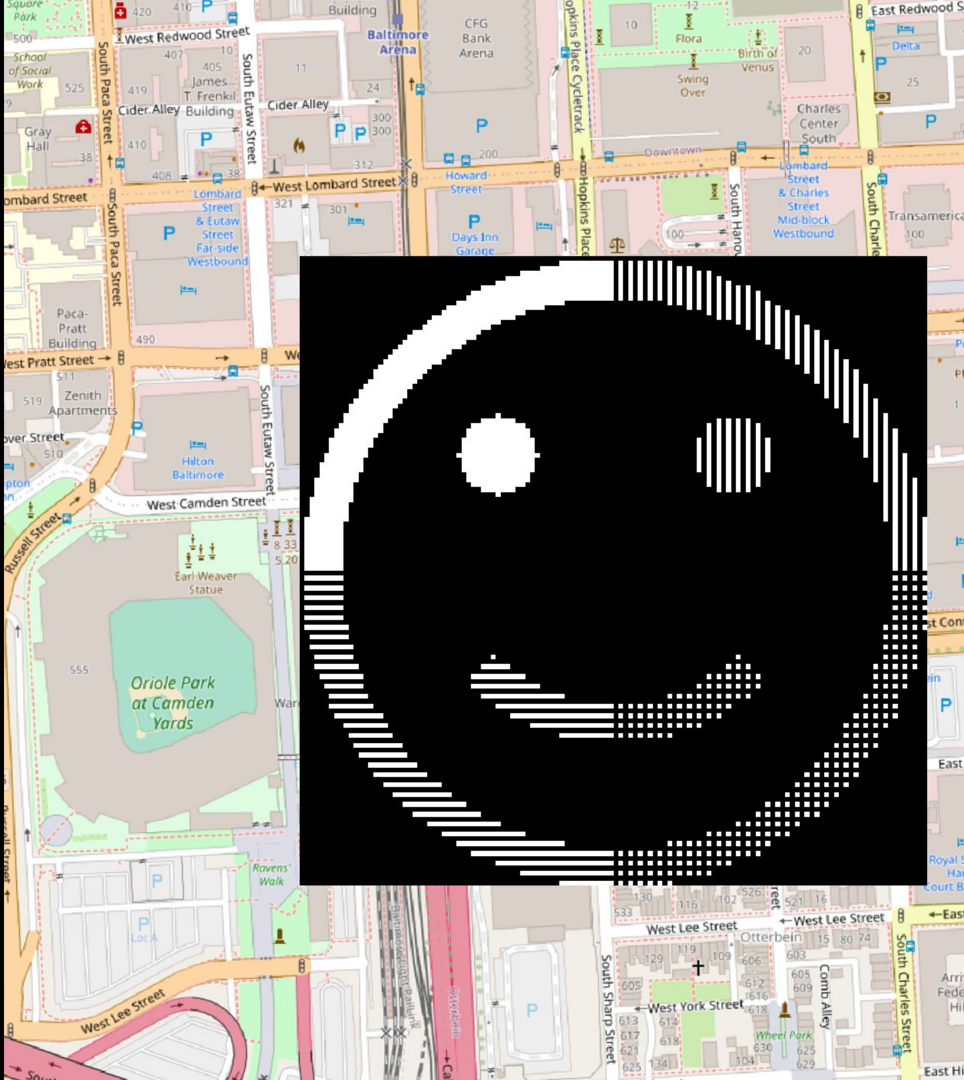
Coordinates of Upper Left and Lower Right corners, in the assigned CRS.

```
smiley.tif
```



Metadata from gdalinfo

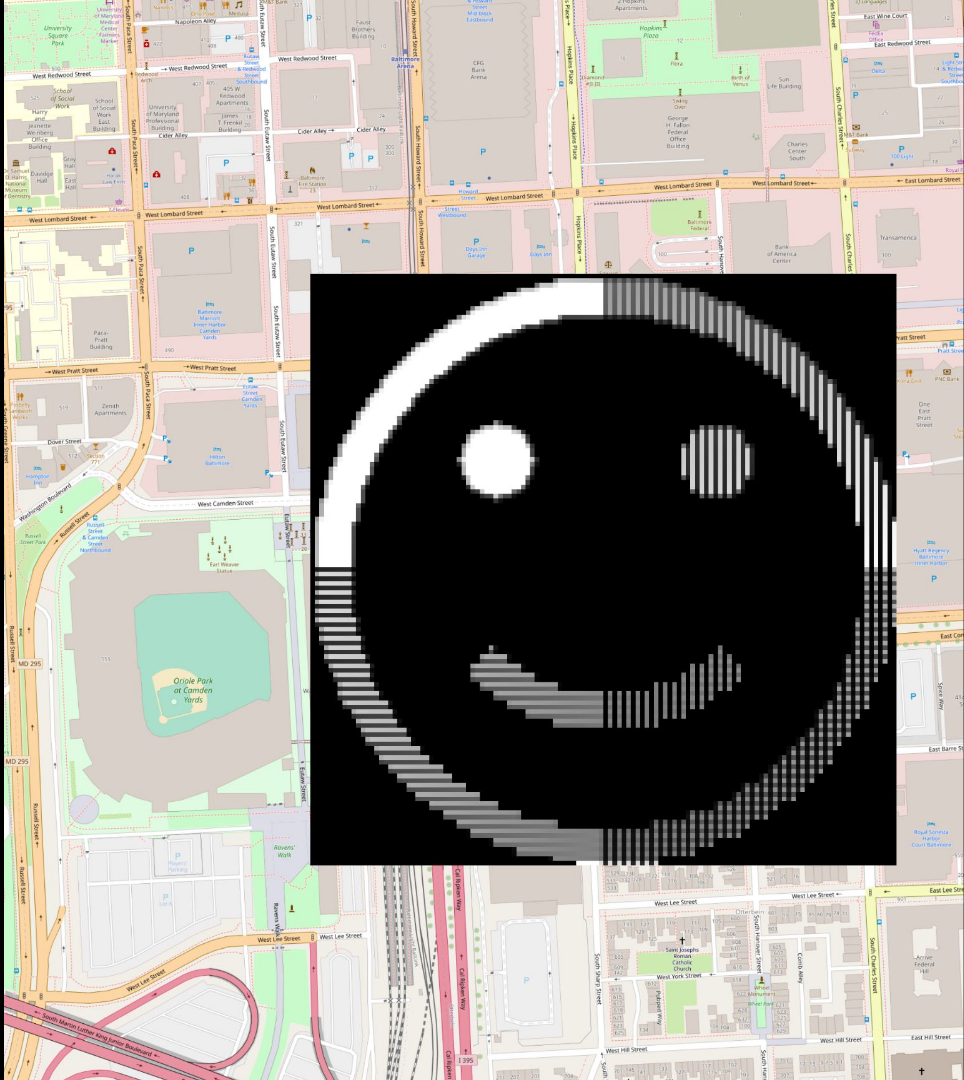
```
$ gdalinfo smiley.tif

Driver: GTiff/GeoTIFF

Size is 128, 128

Coordinate System is:
PROJCRS["NAD83 / Maryland",

Origin = (432750.000,179936.000)

Pixel Size = (4.000,-4.000)
```



Warp to Web Mercator

```
$ gdalwarp
-t_srs epsg:3857
-r bilinear
smiley.tif smiley_merc.tif
```

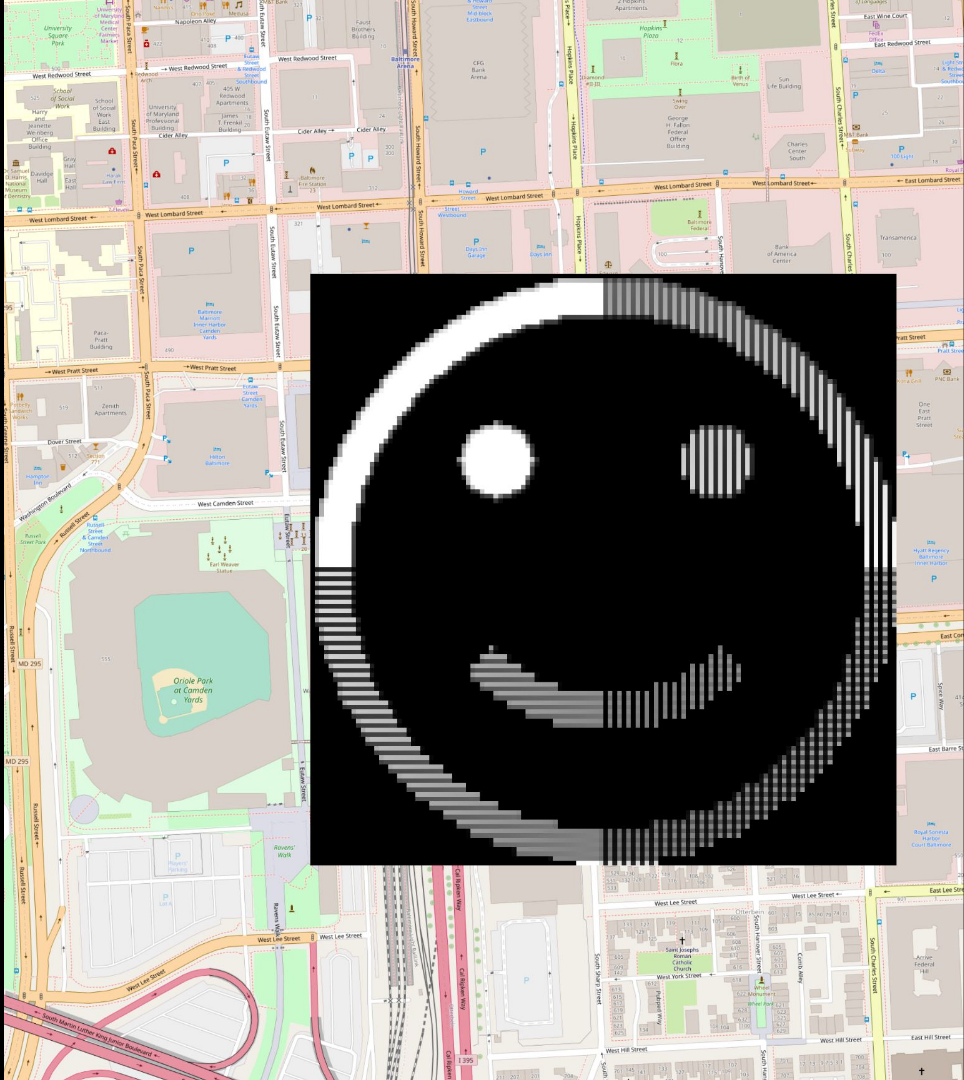
Web Mercator



Warp to Web Mercator

```
$ gdalwarp
-t_srs epsg:3857
-r bilinear
smiley.tif smiley_merc.tif

$ gdalinfo smiley.tif
Driver: GTiff/GeoTIFF
Size is 128, 129
Coordinate System is:
PROJCRS["WGS 84 / Pseudo-Mercator",
Origin = (-8529345.538,476286[illegible].556)
Pixel Size = (5.17160,-5.17160)
```

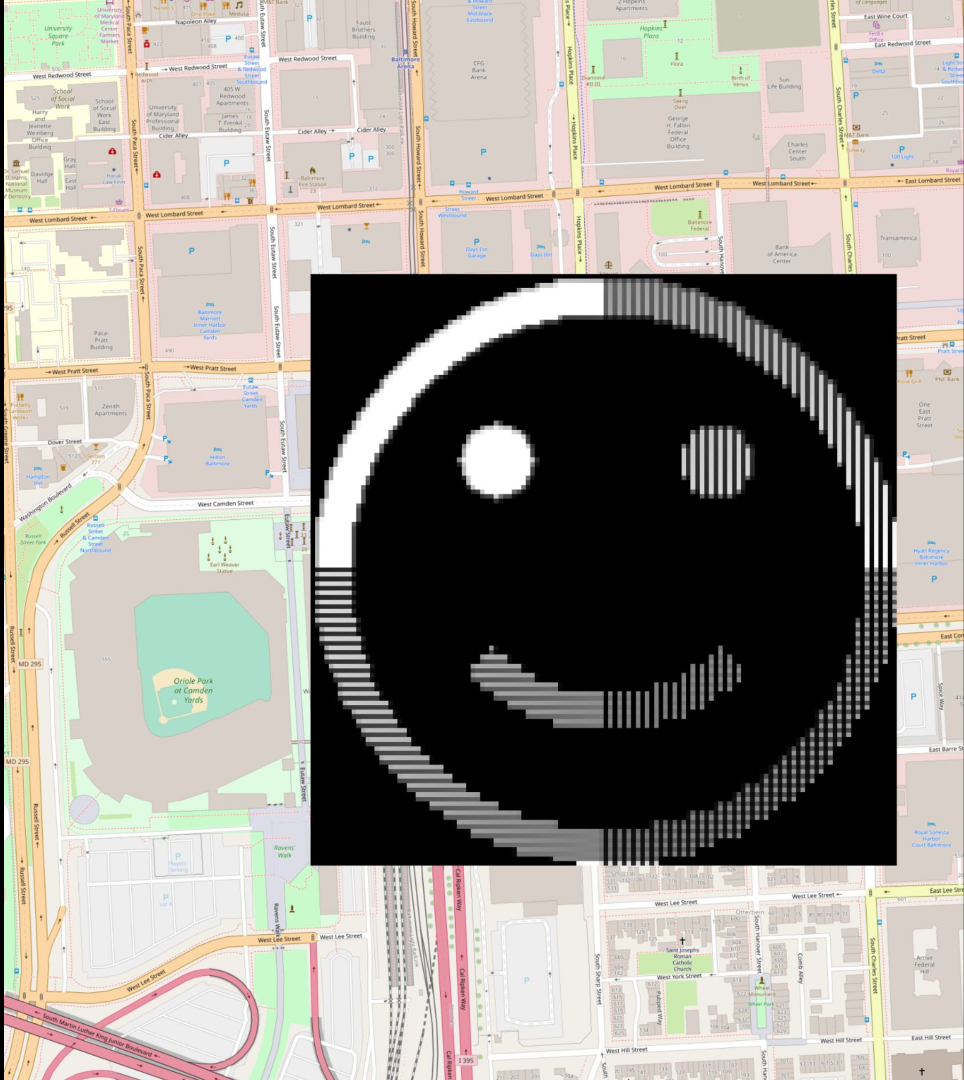
Mercator increases height by 1px, increases pixel size by 1/cos(lat)



Warp to Web Mercator

```
$ gdalwarp
-t_srs epsg:3857
-r bilinear
smiley.tif smiley_merc.tif

$ gdalinfo smiley.tif
Driver: GTiff/GeoTIFF
Size is 128, 129
Coordinate System is:
PROJCRS["WGS 84 / Pseudo-Mercator",
Origin = (-8529345.538,4762866.556)
Pixel Size = (5.17160,-5.17160)
```

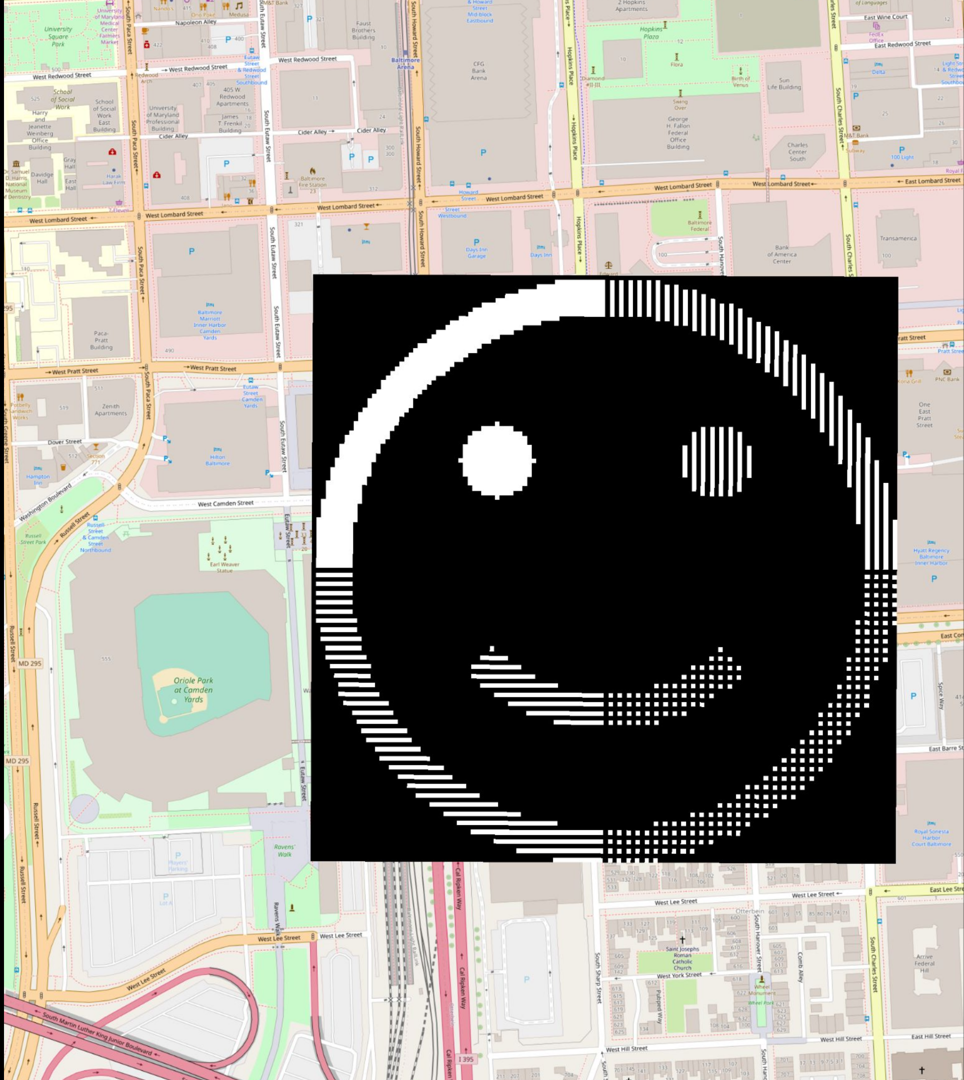
Part of the "geotransform"



GDAL geotransform

$$\begin{bmatrix} x \\ y \end{bmatrix}$$

Georeferenced
(x,y) coordinates

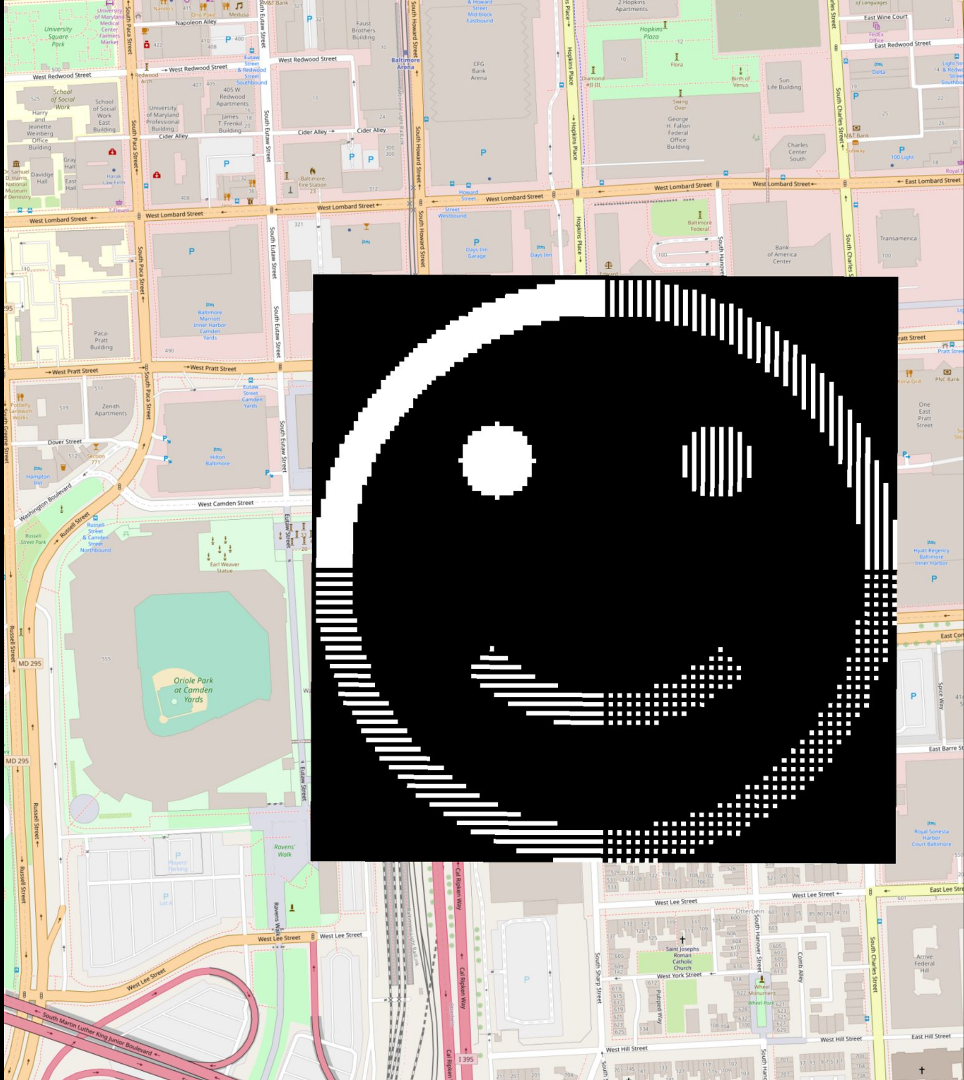


GDAL geotransform

$$\begin{bmatrix} c \\ r \\ 1 \end{bmatrix} \qquad \begin{bmatrix} x \\ y \end{bmatrix}$$

Pixel (column,row) coordinates.

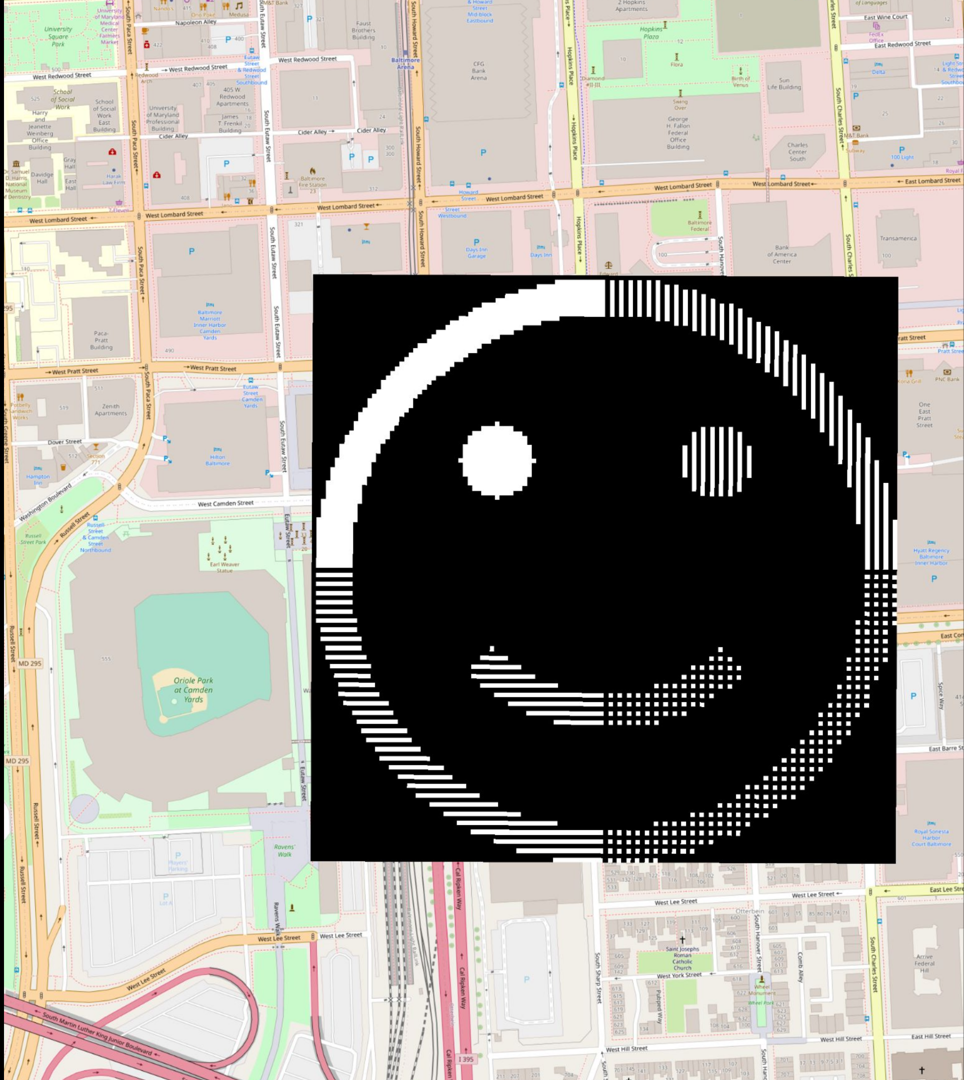
Append 1 to make it "homogeneous"



GDAL geotransform

$$\begin{bmatrix} 4.0 & 0.0 & 432750.0 \\ 0.0 & -4.0 & 179936.0 \end{bmatrix} \times \begin{bmatrix} c \\ r \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix}$$

Geotransform:
a 2D affine transformation.

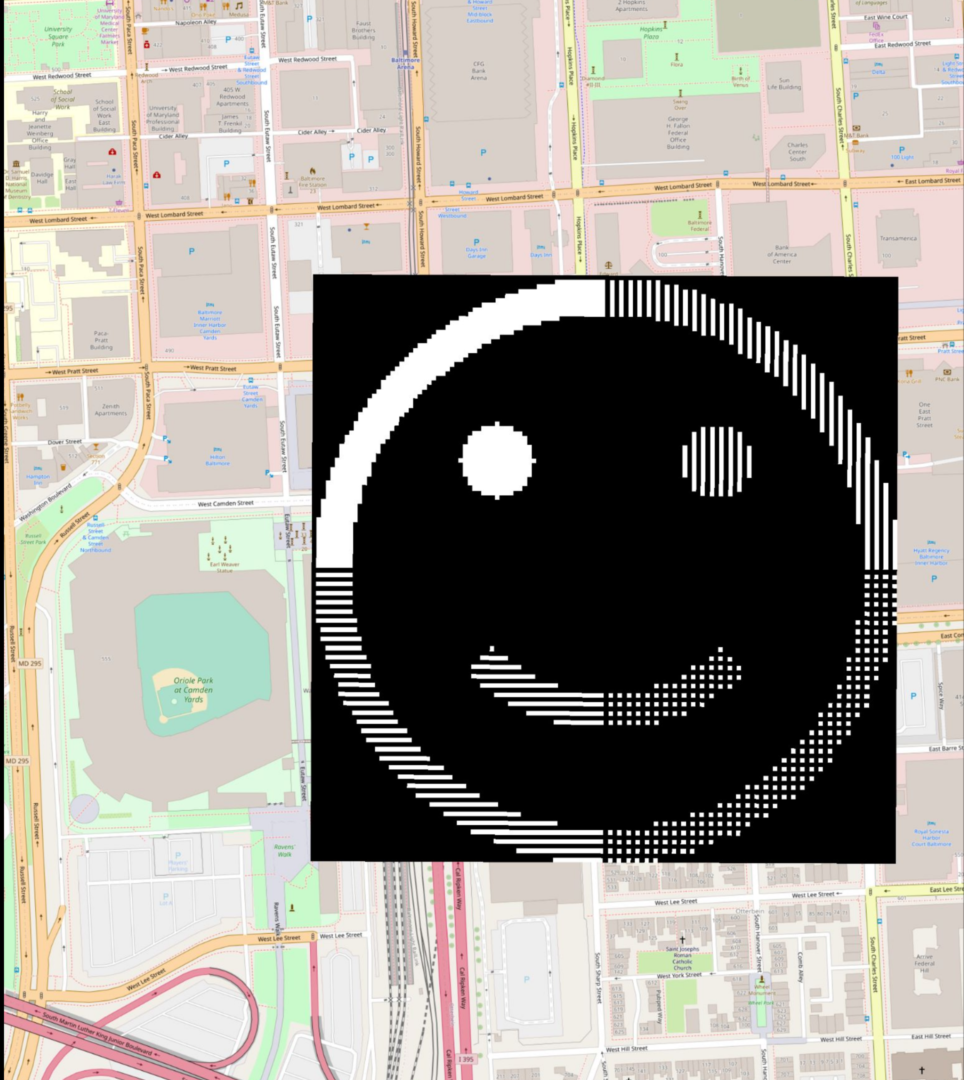


GDAL geotransform

$$\begin{bmatrix} 4.0 & 0.0 & 432750.0 \\ 0.0 & -4.0 & 179936.0 \end{bmatrix} \times \begin{bmatrix} c \\ r \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix}$$

Geotransform:
a 2D affine transformation.

```
Upper-left corner: (c,r) = (0,0)
X: 4*0 +  0*0 + 432750*1 = 432750
Y: 0*0 + -4*0 + 179936*1 = 179936

Bottom-right corner: (c,r) = (128,128)
X: 4*128 +  0*128 + 432750*1 = 433262
Y: 0*128 + -4*128 + 179936*1 = 179424
```

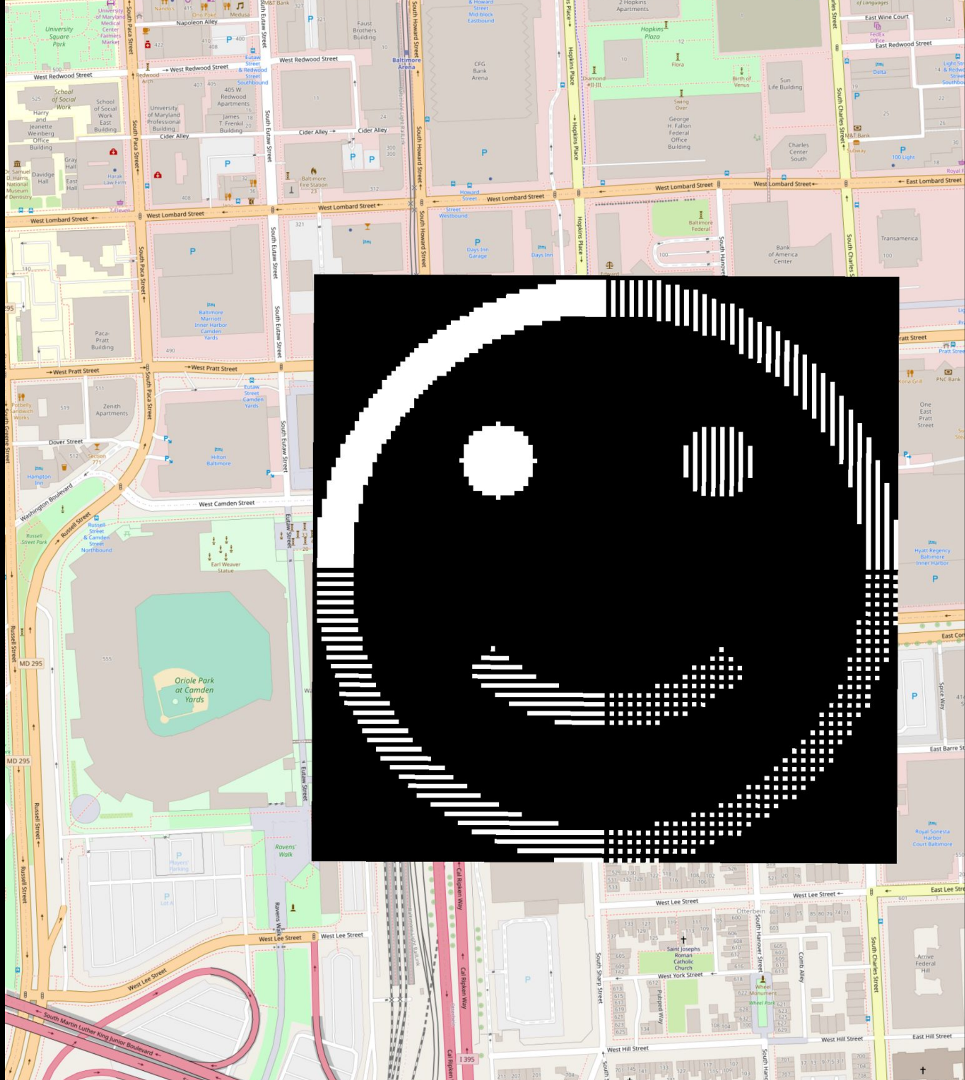


GDAL geotransform

$$\begin{bmatrix} 4.0 & 0.0 & 432750.0 \\ 0.0 & -4.0 & 179936.0 \end{bmatrix} \times \begin{bmatrix} c \\ r \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix}$$

Geotransform plus coordinate system metadata (e.g. WKT) are everything QGIS needs to properly geolocate data.

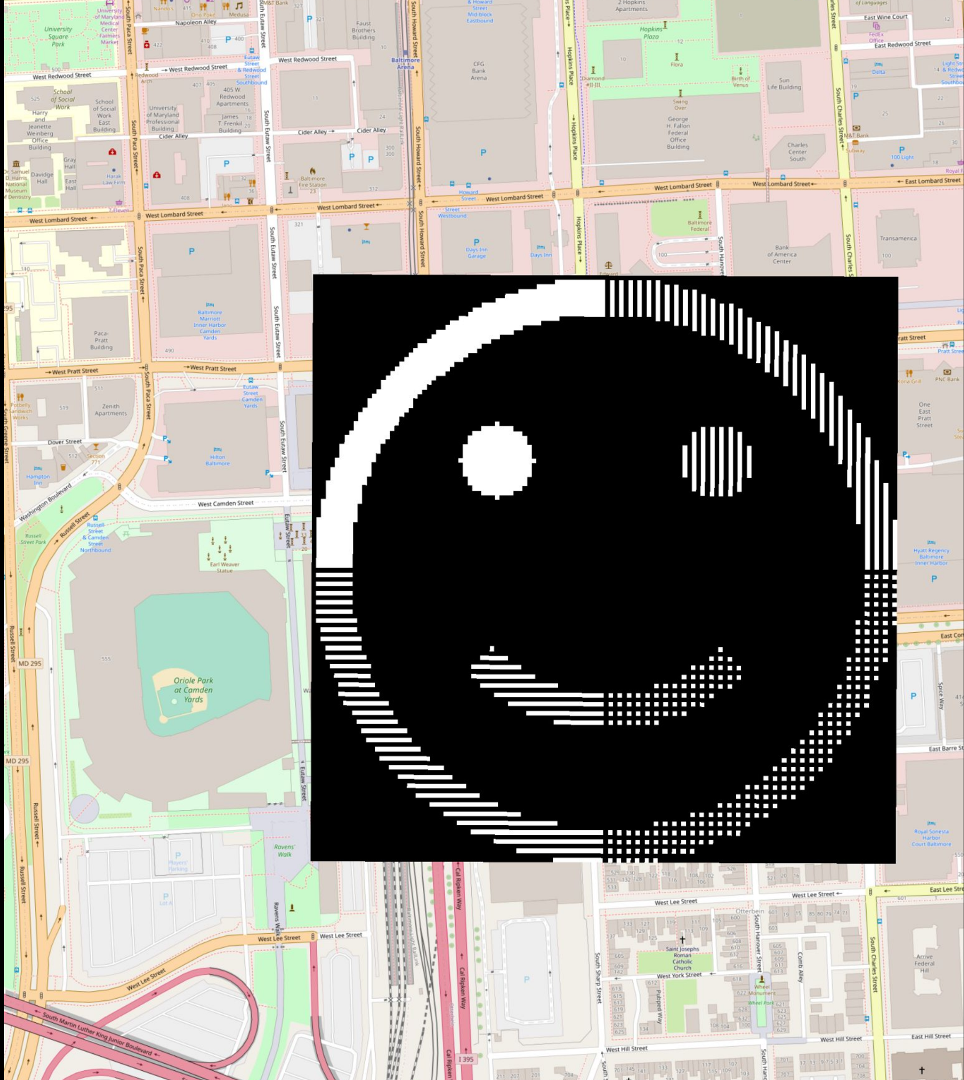
Geotransform parameters are the values stored in the sidecar World File e.g. .tfw, .jgw, .pgw.



Geotransform scale + shift

$$\begin{bmatrix} 4.0 & 0.0 & 432750.0 \\ 0.0 & -4.0 & 179936.0 \end{bmatrix} \times \begin{bmatrix} c \\ r \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix}$$

1. Copy file

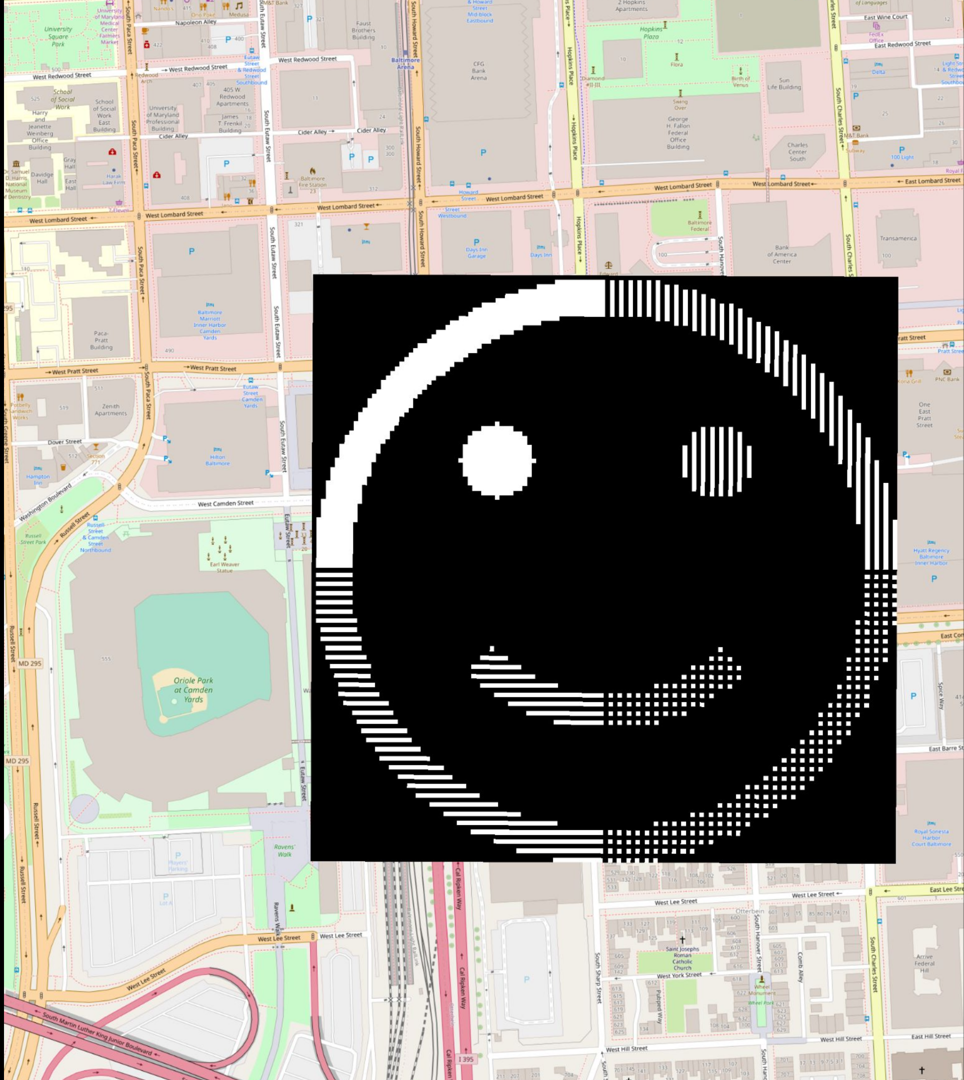


Geotransform scale + shift

$$\begin{bmatrix}4.0 & 0.0 & 432750.0\\ 0.0 & -4.0 & 179936.0\end{bmatrix} \times \begin{bmatrix}c\\ r\\ 1\end{bmatrix} = \begin{bmatrix}x\\ y\end{bmatrix}$$

1. Copy file
2. Transform origin from state plane to Web Mercator via PROJ (or PyProj)

```
x1, y1 = 4332750.0, 179936.0
x2, y2 = pyproj.Transformer.from_crs(
  26985, 3857, always_xy=True
).transform(x1, y1)
```

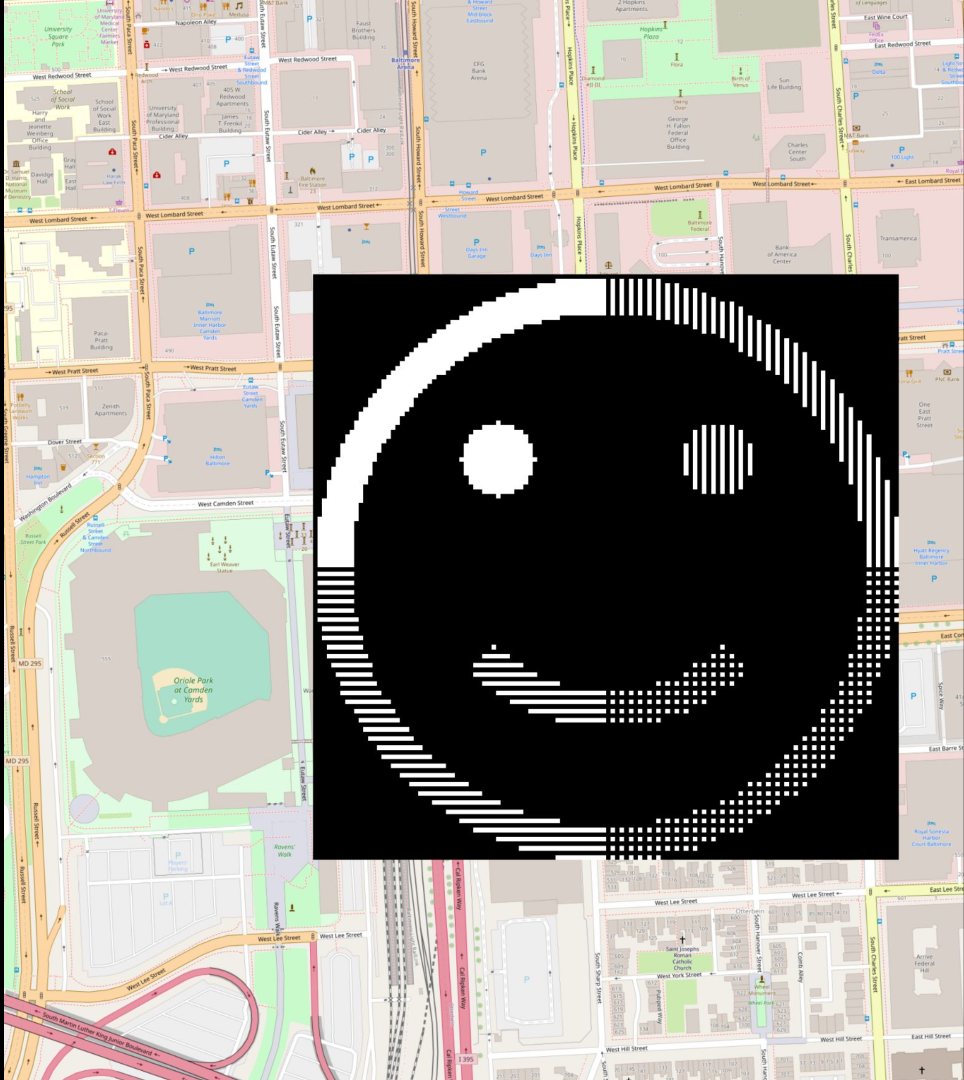


Geotransform scale + shift

$$\begin{bmatrix} \boxed{4.0} & 0.0 & \boxed{432750.0} \\ 0.0 & \boxed{-4.0} & 179936.0 \end{bmatrix} \times \begin{bmatrix} c \\ r \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix}$$

1. Copy file
2. Transform origin from state plane to Web Mercator via PROJ (or PyProj)
3. Scale cell size by Mercator scale factor

```
x1, y1 = 4332750.0, 179936.0
lng, lat = pyproj.Transformer.from_crs(
  26985, 4326, always_xy=True
).transform(x1, y1)
size = 1 / np.cos(np.radians(lat)) * 4.0
```

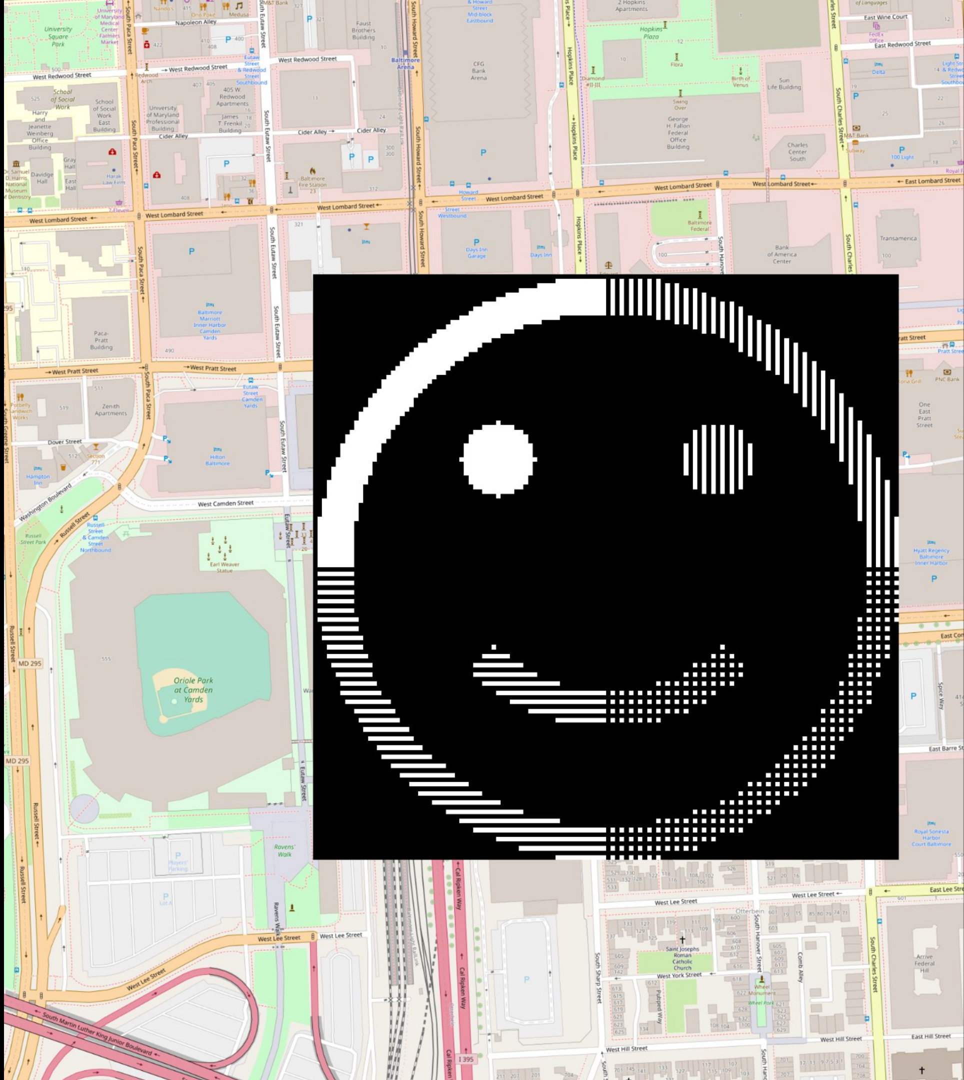


Geotransform scale + shift

$$\begin{bmatrix} \boxed{4.0} & 0.0 & \boxed{432750.0} \\ 0.0 & \boxed{-4.0} & 179936.0 \end{bmatrix} \times \begin{bmatrix} c \\ r \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix}$$

1. Copy file
2. Transform origin from state plane to Web Mercator via PROJ (or PyProj)
3. Scale cell size by Mercator scale factor
4. Update geotransform and CRS in-place.

```
dataset: gdal.Dataset = gdal.Open(
  'smiley_merc.tif', gdal.GA_Update
)
geotransform = (x2, size, 0, y2, 0, -size)
dataset.SetGeoTransform(geotransform)
dataset.SetProjection(pyproj.CRS(3857).to_wkt())
dataset.FlushCache()
dataset = None
```

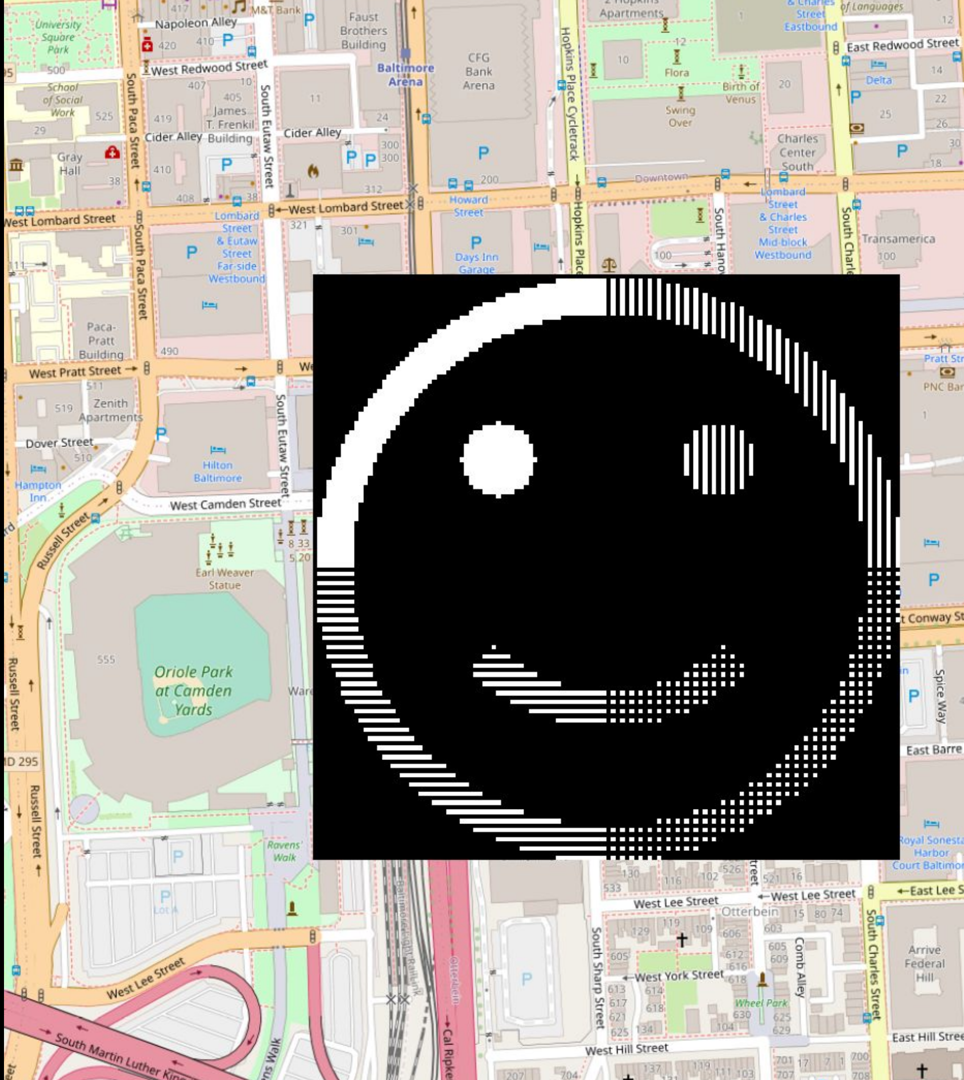


Geotransform scale + shift

$$\begin{bmatrix} \boxed{4.0} & 0.0 & \boxed{432750.0} \\ 0.0 & \boxed{-4.0} & 179936.0 \end{bmatrix} \times \begin{bmatrix} c \\ r \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix}$$

1. Copy file
2. Transform origin from state plane to Web Mercator via PROJ (or PyProj)
3. Scale cell size by Mercator scale factor
4. Update geotransform and CRS in-place.

$$\begin{bmatrix} \boxed{5.1680} & 0.0 & \boxed{-8529342.644} \\ 0.0 & \boxed{-5.1680} & 4762866.676 \end{bmatrix}$$



Geotransform scale + shift

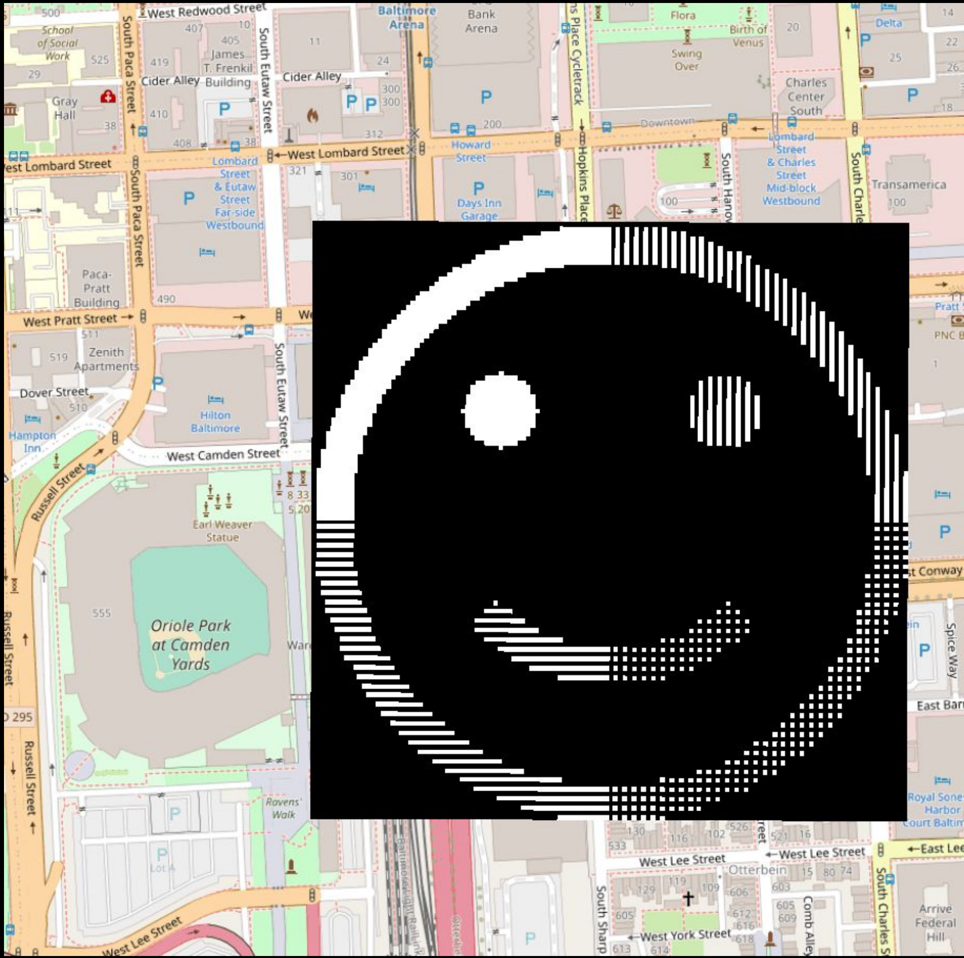
Maximum error:

5.657 meters, 109.5% of a pixel

Root mean squared error:

3.09 meters, 59.8% of a pixel

$$\begin{bmatrix} 5.1680 & 0.0 & -8529342.644 \\ 0.0 & -5.1680 & 4762866.676 \end{bmatrix}$$



Solving for best fit geotransform: derivation

We seek the geotransform aka affine transformation (a,b,c,d,e,f) that best transforms a set of pixel coordinates (c,r) to corresponding georeferenced coordinates (x,y) in a least squares sense.

$$\begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix} \times \begin{bmatrix} c_i \\ r_i \\ 1 \end{bmatrix} = \begin{bmatrix} x_i \\ y_i \end{bmatrix}$$

numpy.linalg.lstsq solver works for systems of equations in the form A*x=b.

Solving for best fit geotransform: derivation

We seek the geotransform aka affine transformation (a,b,c,d,e,f) that best transforms a set of pixel coordinates (c,r) to corresponding georeferenced coordinates (x,y) in a least squares sense.

$$\begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix} \times \begin{bmatrix} c_i \\ r_i \\ 1 \end{bmatrix} = \begin{bmatrix} x_i \\ y_i \end{bmatrix}$$

numpy.linalg.lstsq solver works for systems of equations in the form A*x=b.

$$\overset{\textbf{A}}{\begin{bmatrix} c_1 & r_1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & c_1 & r_1 & 1 \\ c_2 & r_2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & c_2 & r_2 & 1 \\ & \vdots & & & \vdots & \\ c_n & r_n & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & c_n & r_n & 1 \end{bmatrix}} \times \overset{\textbf{x}}{\begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \end{bmatrix}} = \overset{\textbf{b}}{\begin{bmatrix} x_1 \\ y_2 \\ x_2 \\ y_2 \\ \vdots \\ x_n \\ y_n \end{bmatrix}}$$

A is a 2n x 6 matrix with n homogeneous pixel coordinates.
x is the 6 x 1 vector that we're solving for
b is a 2n x 1 vector with n georeferenced coordinates.

Solving for best fit geotransform: implementation

```python
def from_points(filename: Path, src_pts: np.ndarray, dst_pts: np.ndarray, wkt: str):
  matrix_a = []
  vec_b = []
  for src_pt, dst_pt in zip(src_pts, dst_pts):
    matrix_a.append([src_pt[0], src_pt[1], 1, 0, 0, 0])
    matrix_a.append([0, 0, 0, src_pt[0], src_pt[1], 1])
    vec_b.extend([dst_pt[0], dst_pt[1]])
  vec_x = np.linalg.lstsq(np.array(matrix_a), np.array(vec_b), rcond=None)[0]

  # GDAL ordering is different than affine transformation ordering
  geotransform = (vec_x[2], vec_x[0], vec_x[1], vec_x[5], vec_x[3], vec_x[4])
```

Solving for best fit geotransform: implementation

```
def from_points(filename: Path, src_pts: np.ndarray, dst_pts: np.ndarray, wkt: str):
  matrix_a = []
  vec_b = []
  for src_pt, dst_pt in zip(src_pts, dst_pts):
    matrix_a.append([src_pt[0], src_pt[1], 1, 0, 0, 0])
    matrix_a.append([0, 0, 0, src_pt[0], src_pt[1], 1])
    vec_b.extend([dst_pt[0], dst_pt[1]])
  vec_x = np.linalg.lstsq(np.array(matrix_a), np.array(vec_b), rcond=None)[0]

  # GDAL ordering is different than affine transformation ordering
  geotransform = (vec_x[2], vec_x[0], vec_x[1], vec_x[5], vec_x[3], vec_x[4])

  dataset: gdal.Dataset = gdal.Open(str(filename), gdal.GA_Update)
  dataset.SetGeoTransform(geotransform)
  dataset.SetProjection(wkt)
  dataset.FlushCache()
  dataset = None
```

Geotransform best fit

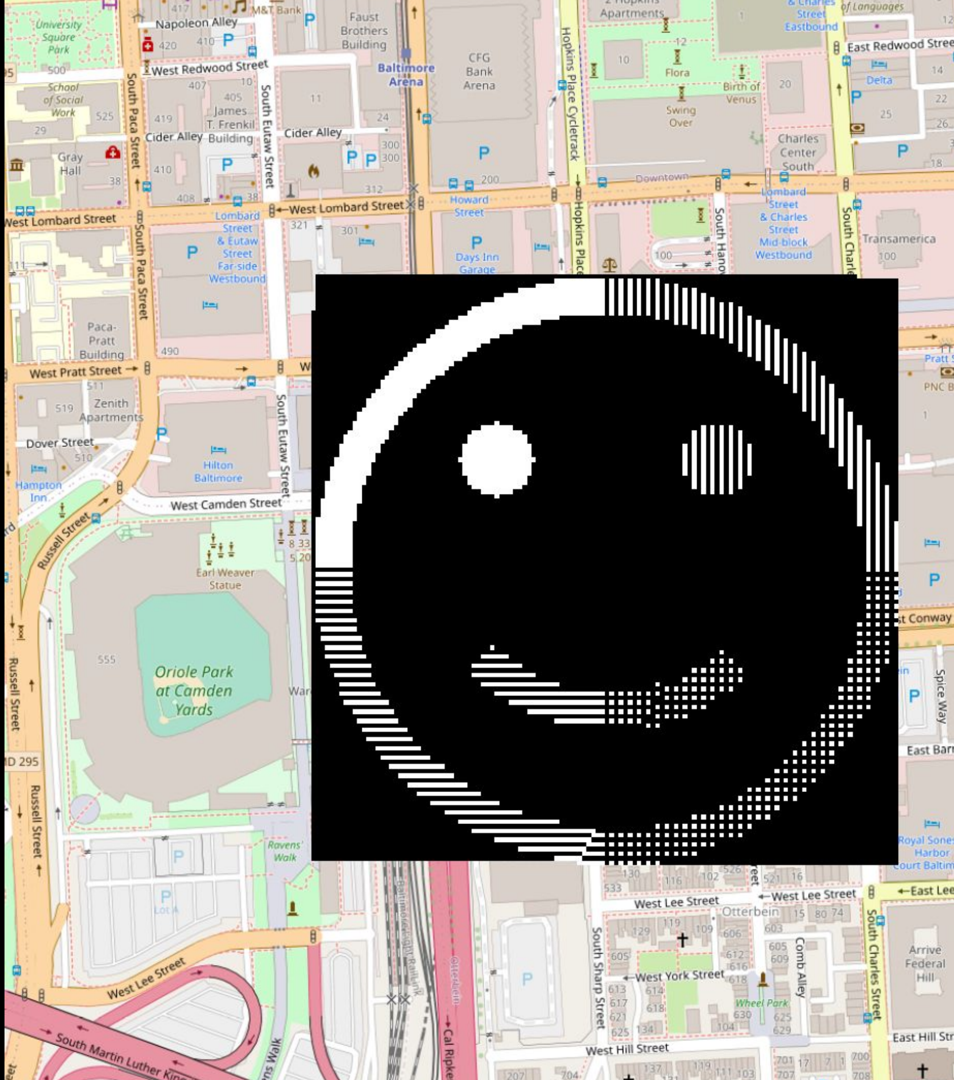
$$\begin{bmatrix} 5.1610 & -0.0216 & -8529342.634 \\ -0.0216 & -5.1818 & 4762866.676 \end{bmatrix}$$

Maximum error:

0.010 meters, 0.195% of a pixel

Root mean squared error:

0.004 meters, 0.082% of a pixel



Geotransform best fit

1 Answer

Sorted by: Highest score (default)

1

To my understanding this is actually an issue with QGIS, and there is an open issue related to providing better support for rotated rasters.

See also: https://github.com/qgis/QGIS/issues/22012

GDAL *does* consider the pixel grid as being truly rotated. The geotransform provides an affine transformation from image to map space, so if the geotransform describes a rotation then a grid in image space would obviously end up rotated in map space. The geotransform tutorial gives a good overview.

Both the unrotated pixels and the apparent resampling are display artifacts, and don't reflect the underlying data.

Share Edit Follow Flag

edited Sep 22, 2022 at 17:01

answered Sep 22, 2022 at 3:46

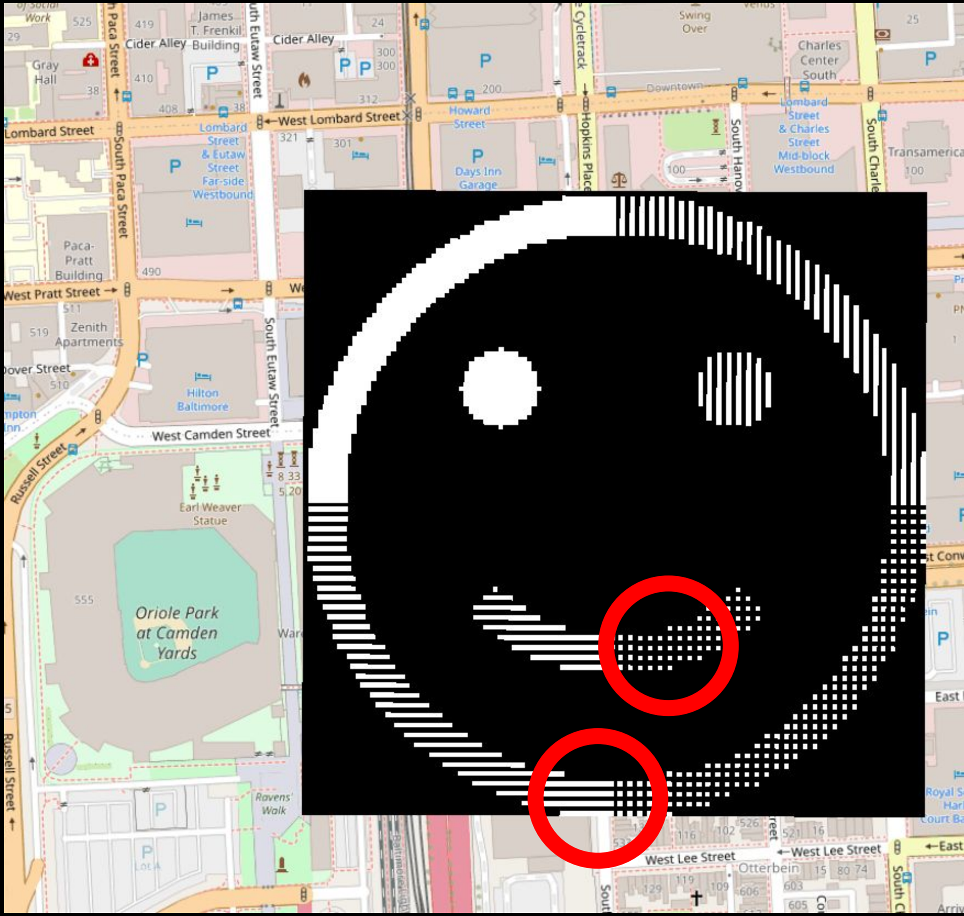
corvus

526 6 18

bheberlein commented on Nov 4, 2022

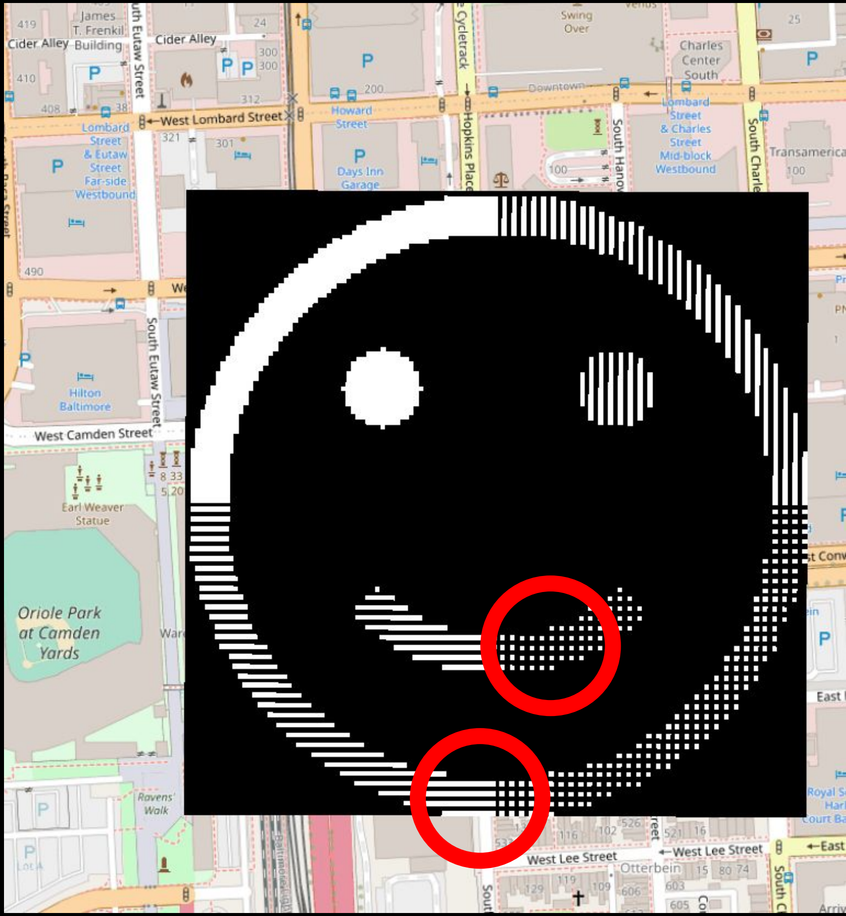
I would request this issue be re-opened, #46794 only partially fixes it. The pixels of a rotated grid still are displayed North-up, and the image data itself is resampled for display which is confusing and (I think) surely not the desired behavior in the majority of use cases.

1



Geotransform best fit

ArcGIS Pro: OK



QGIS: not OK

Geotransform best fit, no rotation allowed

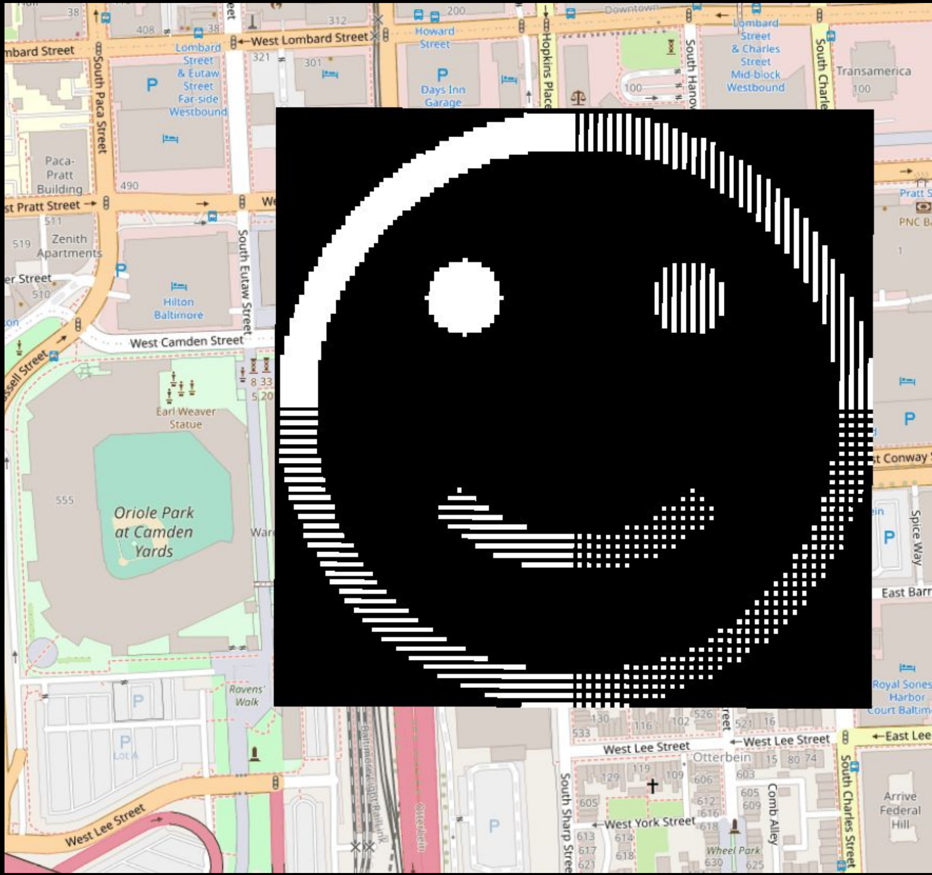
$$\begin{bmatrix} 5.1610 & 0.0 & -8529343.974 \\ 0.0 & -5.181 & 4762865.331 \end{bmatrix}$$

Maximum error:

1.906 meters, 36.85% of a pixel

Root mean squared error:

1.131 meters, 21.87% of a pixel



Error comparison for various CRSs

EPSG code	3857
CRS name	Web Mercator (auxiliary sphere)
Max error: best fit w/ rotation	0.01 m
	0.195% of pixel
Max error: best fit w/o rotation	1.90 m
	36.8% of pixel

Error comparison for various CRSs

EPSG code	3857	32618
CRS name	Web Mercator (auxiliary sphere)	WGS84 / UTM Zone 18N
Max error: best fit w/ rotation	0.01 m	0.0002 m
	0.195% of pixel	0.005% of pixel
Max error: best fit w/o rotation	1.90 m	7.737 m
	36.8% of pixel	193% of pixel

Error comparison for various CRSs

EPSG code	3857	32618	5070
CRS name	Web Mercator (auxiliary sphere)	WGS84 / UTM Zone 18N	NAD83 / Conus Albers
Max error: best fit w/ rotation	0.01 m	0.0002 m	0.0003 m
	0.195% of pixel	0.005% of pixel	0.008% of pixel
Max error: best fit w/o rotation	1.90 m	7.737 m	69.6 m
	36.8% of pixel	193% of pixel	1775% of pixel

Conclusions

Resampling during gdalwarp almost always degrades image quality, even subtly.

Many canonical gdalwarp transformations can be approximated with an affine transformation to within a fraction of a pixel. This avoids:

- A lossy resample
- gdalwarp's processing time

Conclusions

Resampling during gdalwarp almost always degrades image quality, even subtly.

Many canonical gdalwarp transformations can be approximated with an affine transformation to within a fraction of a pixel. This avoids:

- A lossy resample
- gdalwarp's processing time

But this approximation doesn't always work:

- Large extents, e.g. global scale, won't fit.
- Unusual transformations might not fit.
- Even small georeferencing errors might not be tolerable in some situations.

If viewing in QGIS, you must settle for either:

- Rendering artifacts due to rotated pixel bug
- A less accurate affine transformation that only performs translation and scaling but does not perform a rotation.

Conclusions

Resampling during gdalwarp almost always degrades image quality, even subtly.

Many canonical gdalwarp transformations can be approximated with an affine transformation to within a fraction of a pixel. This avoids:

- A lossy resample
- gdalwarp's processing time

But this approximation doesn't always work:

- Large extents, e.g. global scale, won't fit.
- Unusual transformations might not fit.
- Even small georeferencing errors might not be tolerable in some situations.

If viewing in QGIS, you must settle for either:

- Rendering artifacts due to rotated pixel bug
- A less accurate affine transformation that only performs translation and scaling but does not perform a rotation.

Give it a shot! Worst case you fall back on gdalwarp, best case you avoid resampling and waiting.