Art and Science With Six Million Timelapse Images and Python

Eric Floehr
eric@intellovations.com

Talk and code at:

https://github.com/efloehr/timelapse

(fair warning: it's evolving!)

Camera Setup

Timelapse Equipment

- Canon A510 digital camera...
- Mounted securely.
- Controlled by gphoto2...
- ...On a Raspberry Pi.
- Connected to a UPS...
- ...And wired network.
- Taking a picture every 10 seconds.
- Permanent storage on RAIDed NAS (Synology)

Looking at Images

I'm on my third version of this!
(and a lot of experimenting)

What I Learned From Experimenting

- SD cards are unreliable
  - Gphoto2 can pull directly from camera memory
  - Create a tmpfs RAM disk to store on RPi
  - Move to real disk quickly
- Create buffers where possible
- Rely on as little as possible
  - RPi and NAS connected to switch
  - Static IP addresses

Where are we now?

- Started up July 3, 2013
- 6,434,650 images (and counting)
- 20,241,658,675,200 pixels (20 trillion)
- 4.2 terabytes of storage
- On third camera
- First camera lasted 319 days and 2.7 million pics
- On second Raspberry Pi
- That's a LOT of data

We Need A Way To Organize

We need a way to organize

- Store image data in PostgreSQL
- Use Django ORM (yes Django!)
  - Gives us Python models, easy queries, migrations
- Pull metadata with Python, Pillow, and pyxif
  - Time photo was taken
  - Exposure and f/stop
  - Size
  - Some color information about the sky

Image Info Model

```
class Info(models.Model):
    # File info
    filepath = models.CharField(max_length=1024, unique=True)
    filename = models.CharField(max_length=255, unique=True, null=True)
    size = models.IntegerField(default=0)
    timestamp = models.DateTimeField(db_index=True, unique=True)

    # Exif Info
    fstop = models.IntegerField(null=True) # * 100
    exposure = models.IntegerField(null=True) # in microseconds

    # Color Info (on clouds only)
    center_color = models.IntegerField(null=True)
    ...
    stddev_blue = models.IntegerField(null=True)

    min_color = models.IntegerField(null=True)
    max_color = models.IntegerField(null=True)

    class Meta:
        ordering = ['timestamp']
```

Image Info Model

```
class Info(models.Model):
    # File info
    filepath = models.CharField(max_length=1024, unique=True)
    filename = models.CharField(max_length=255, unique=True, null=True)
    size = models.IntegerField(default=0)
    timestamp = models.DateTimeField(db_index=True, unique=True)

    # Exif Info
    fstop = models.IntegerField(null=True) # * 100
    exposure = models.IntegerField(null=True) # in microseconds

    # Color Info (on clouds only)
    center_color = models.IntegerField(null=True)
    ...
    stddev_blue = models.IntegerField(null=True)

    min_color = models.IntegerField(null=True)
    max_color = models.IntegerField(null=True)

    class Meta:
        ordering = ['timestamp']
```

Image Info Model

```
class Info(models.Model):
    # File info
    filepath = models.CharField(max_length=1024, unique=True)
    filename = models.CharField(max_length=255, unique=True, null=True)
    size = models.IntegerField(default=0)
    timestamp = models.DateTimeField(db_index=True, unique=True)

    # Exif Info
    fstop = models.IntegerField(null=True) # * 100
    exposure = models.IntegerField(null=True) # in microseconds

    # Color Info (on clouds only)
    center_color = models.IntegerField(null=True)
    ...
    stddev_blue = models.IntegerField(null=True)

    min_color = models.IntegerField(null=True)
    max_color = models.IntegerField(null=True)

    class Meta:
        ordering = ['timestamp']
```

Image Info Model

```
class Info(models.Model):
    # File info
    filepath = models.CharField(max_length=1024, unique=True)
    filename = models.CharField(max_length=255, unique=True, null=True)
    size = models.IntegerField(default=0)
    timestamp = models.DateTimeField(db_index=True, unique=True)

    # Exif Info
    fstop = models.IntegerField(null=True) # * 100
    exposure = models.IntegerField(null=True) # in microseconds

    # Color Info (on clouds only)
    center_color = models.IntegerField(null=True)
    ...
    stddev_blue = models.IntegerField(null=True)

    min_color = models.IntegerField(null=True)
    max_color = models.IntegerField(null=True)

    class Meta:
        ordering = ['timestamp']
```

Inserting Image Data

```
@classmethod
def insert(cls, filepath, check_for_existing=True):
    ...
    # Load image, exif, and file information
    im = Image.open(filepath)
    dummy, exif_dict, dummy = pyxif.load(filepath)
    filestat = os.stat(filepath)

    # File stats
    pic.filename = os.path.basename(filepath)
    pic.size = filestat.st_size

    # Exif Info
    pic.fstop, pic.exposure = get_fstop_exposure(exif_dict)

    # Color info (Clouds only)
    imclouds = im.crop((0,top_row,width,bottom_row))
    cloudstats = ImageStat.Stat(imclouds)
    exc = cloudstats.extrema

    pic.mean_color = rgb_to_int(cloudstats.mean)
```

Let's Explore The Data

Larger Images are More Complex

```
=> select filepath, size from image_info order by size desc limit 1;

                          filepath                                  |  size
--------------------------------------------------------------------+---------
 /var/timelapse/2013-10-16/22/2013-10-16-22-23-42.jpg | 1256651
(1 row)
```

More Color Deviation = Interesting

```
=> select filepath, stddev_red + stddev_green + stddev_blue as stddev
   from image_info order by stddev desc limit 1;

                          filepath                          | stddev
------------------------------------------------------------+--------
 /var/timelapse/2014-07-08/22/2014-07-08-22-40-51.jpg |    227
(1 row)
```

Smoothing Out The Imperfections (Normalization)

Normalization

- Images don't always align on 10 second boundaries
- Allow for gaps in the data
- Provides for a placeholder
- Match up same time in different days easily
- Makes queries much easier

Normalization

- Start with the closest normalized time match to the image time
- If that normal already has an image, put the new image in the next normal slot as long as the time delta is less than 10 seconds, otherwise discard
- Keep just filling normal slots unless the time delta between normal and image is greater than 10 seconds
- Once it's greater than 10 seconds, renormalize to closest match (step one above)
- This prevents the flapping problem of always using the closest normal, which can cause gaps when the time between images isn't exactly 10 seconds and oscillates between the midpoint between normals

Normalized Model

```
class Normal(models.Model):
    timestamp = models.DateTimeField(db_index=True)
    info = models.ForeignKey(Info, null=True)

    SECONDS_BASE = 10

    class Meta:
        ordering = ['timestamp']
```

Creating Normals

```
def normalize_time(timestamp, seconds_base=10):
    # Normalize to seconds_base seconds
    normal_timestamp = timestamp
    # Round works differently in Python 3 than 2
    start_second = seconds_base * int(math.floor(timestamp.second /
                                      float(seconds_base) + 0.5))
    if start_second >= 60:
        normalized_timestamp = timestamp + timedelta(minutes=1)
        start_second = 0

    normal_timestamp = normal_timestamp.replace(second=start_second)
    normal_timestamp = normal_timestamp.replace(microsecond=0)

    return normalized_timestamp
```

Creating Normals

```
# First normal want to get as close as possible
normal_image_tstamp = normalize_time(current_image.timestamp, 10)
```

```
# Now just keep going and don't skip until we have a delta > 10
normal_iterator = iter(normals[1:])
for normal in normal_iterator:
    diff_seconds = (current_image.timestamp – normal.timestamp)
    diff_seconds = diff_seconds.total_seconds()

    if abs(diff_seconds) <= 10:
        # That's exactly what we want
        pass
    else:
        if current_image.timestamp < normal.timestamp:
            # Walk up the image to the normal
            # Now we may need to walk the normal to the image
            # Walk up the normal to the image
```

Creating Normals

```
# First normal want to get as close as possible
normal_image_tstamp = normalize_time(current_image.timestamp, 10)

# Now just keep going and don't skip until we have a delta > 10
normal_iterator = iter(normals[1:])
for normal in normal_iterator:
    diff_seconds = (current_image.timestamp - normal.timestamp)
    diff_seconds = diff_seconds.total_seconds()

    if abs(diff_seconds) <= 10:
        # That's exactly what we want
        pass
    else:
        if current_image.timestamp < normal.timestamp:
            # Walk up the image to the normal
            # Now we may need to walk the normal to the image
            # Walk up the normal to the image
```

Creating Normals

```
# First normal want to get as close as possible
normal_image_tstamp = normalize_time(current_image.timestamp, 10)

# Now just keep going and don't skip until we have a delta > 10
normal_iterator = iter(normals[1:])
for normal in normal_iterator:
    diff_seconds = (current_image.timestamp – normal.timestamp)
    diff_seconds = diff_seconds.total_seconds()
```

```
if abs(diff_seconds) <= 10:
    # That's exactly what we want
    pass
else:
    if current_image.timestamp < normal.timestamp:
        # Walk up the image to the normal
        # Now we may need to walk the normal to the image
        # Walk up the normal to the image
```

Making Timelapse Videos

```
mencoder
   -really-quiet
   mf://@imagefilelist.txt
   -mf fps=24:type=jpg
   -ovc x264
   -lavcopts vcodec=mpeg4:mbd=2:trell
   -vf scale=1440:1080
   -oac copy
   -o movie.mp4
```

Making Timelapse Videos (cont.)

```
def make_video(imagelistfile, framerate, moviefilepath):
    subprocess.check_call(['mencoder',
                           '-really-quiet',
                           'mf://@{0}'.format(imagelistfile),
                           '-mf', 'fps={0}:type=jpg'.format(framerate),
                           '-ovc', 'x264',
                           '-lavcopts',
                           'vcodec=mpeg4:mbd=2:trell',
                           '-vf',
                           'scale=1440:1080',
                           '-oac',
                           'copy',
                           '-o',
                           moviefilepath
                           ])
```

Thunderstorm Rolls In Video

An Eternity in an Hour?

- Movie defaults to 24 frames per second
- Each frame is 10 seconds of real time
- 6,527,237 frames (there were a few outages)
- 65 million seconds, or 1.1 million minutes, or 18,000 hours, or 755 days of real time
- Each second of movie is 24 frames representing 240 seconds of real time
- A movie with all the frames would be over 3 days long
- A one hour movie would be 10 days of real time at 24 images per second
- 10 days is like an eternity to my teenage daughters. QED.

Images in Context
(Sunrise, Sunset, Moonrise, etc.)

Using Pyephem

```
def sunset(observer, date):
    observer.date = date.astimezone(utc)
    return observer.next_setting(sun).datetime().replace(tzinfo=utc)
```

```
def sunsets(observer, start_date, offset_seconds=0):
    # offset negative before, positive after
    current_date = copy(start_date)
    while 1:
        yield sunset(observer, current_date) +
                timedelta(seconds=offset_seconds)
        current_date = current_date + timedelta(days=1)
```

Using Generators to Get Sunsets

```
def sunset(observer, date):
    observer.date = date.astimezone(utc)
    return observer.next_setting(sun).datetime().replace(tzinfo=utc)


def sunsets(observer, start_date, offset_seconds=0):
    # offset negative before, positive after
    current_date = copy(start_date)
    while True:
        yield sunset(observer, current_date) +
                timedelta(seconds=offset_seconds)
        current_date = current_date + timedelta(days=1)
```

Using Generators to Get Sunsets

```
def sunset(observer, date):
    observer.date = date.astimezone(utc)
    return observer.next_setting(sun).datetime().replace(tzinfo=utc)


def sunsets(observer, start_date, offset_seconds=0):
    # offset negative before, positive after
    current_date = copy(start_date)
    while True:
        yield sunset(observer, current_date) +
                timedelta(seconds=offset_seconds)
        current_date = current_date + timedelta(days=1)
```

The Progression of Time

June 21, 2013

December 30, 2013

The Progression of Time Video

Ancient Astronomical Alignments

Photo credits:
http://en.wikipedia.org/wiki/File:ChichenItzaEquinox.jpg
http://www.colorado.edu/Conferences/chaco/tour/images/dagger.jpg

Reducing Our Workload (Image Stacking)

You Can't Look At Them All

Looking at one image per second, it would take you two and a half hours every day to review the previous day's images

Reduce our Workload

- At night, interesting things are lighter
- Stack all the images of a night into a single image that identifies all “light” pixels
- Use pillow's ImageChops.lighter() to get the lighter pixel between images
- During the day, interesting things are darker (mostly)
- Use pillow's ImageChops.darker() to get the darker pixel between images

Making an All Night Image

```
@task()
def make_all_night_image(day):
    # Normalize to midnight
    day_start = datetime(day.year, day.month, day.day, tzinfo=est)

    # Find that day's sunset
    obs = get_observer()
    sunset_time = sunset(obs, day_start)

    # And next day's sunrise
    sunrise_time = sunrise(obs, sunset_time)

    # One hour after and before to remove all light
    start_time = normalize_time(sunset_time + timedelta(hours=1))
    end_time = normalize_time(sunrise_time - timedelta(hours=1))

    times = Normal.objects.filter(timestamp__gte=start_time,
                    timestamp__lte=end_time, picture__id__isnull=False)

    img = Image.new("L", (2048,1536), background_color)
```

Making an All Night Image

```
@task()
def make_all_night_image(day):
    # Normalize to midnight
    day_start = datetime(day.year, day.month, day.day, tzinfo=est)

    # Find that day's sunset
    obs = get_observer()
    sunset_time = sunset(obs, day_start)

    # And next day's sunrise
    sunrise_time = sunrise(obs, sunset_time)

    # One hour after and before to remove all light
    start_time = normalize_time(sunset_time + timedelta(hours=1))
    end_time = normalize_time(sunrise_time - timedelta(hours=1))

    times = Normal.objects.filter(timestamp__gte=start_time,
                    timestamp__lte=end_time, picture__id__isnull=False)

    img = Image.new("L", (2048,1536), background_color)
```

Making an All Night Image

```
@task()
def make_all_night_image(day):
    # Normalize to midnight
    day_start = datetime(day.year, day.month, day.day, tzinfo=est)

    # Find that day's sunset
    obs = get_observer()
    sunset_time = sunset(obs, day_start)

    # And next day's sunrise
    sunrise_time = sunrise(obs, sunset_time)

    # One hour after and before to remove all light
    start_time = normalize_time(sunset_time + timedelta(hours=1))
    end_time = normalize_time(sunrise_time - timedelta(hours=1))

    times = Normal.objects.filter(timestamp__gte=start_time,
                    timestamp__lte=end_time, picture__id__isnull=False)

    img = Image.new("L", (2048,1536), background_color)
```

Using PIL's ImageChops

```
 for time in times:
    source = Image.open(time.picture.filepath)

    # Make light image
    if img_light is None:
        img_light = source
    else:
        img_light = ImageChops.lighter(img_light, source)

# Put a date on the image
daystr = day.strftime('%Y-%m-%d')
canvas = ImageDraw.Draw(img_light)
canvas.text((20,1500), daystr)

# And save
img.save(os.path.join(dirpath, filename))
img_light.save(os.path.join(dirpath, filename_light))

# Invert (negative version)
ImageOps.invert(img_light).save(img_neg_filepath)
```

Lightning!

Result (negative version)

Zooming In

More Lightning

More Lightning (Multiple Images)

No UFOs

But lots of planets and stars

Negative version

You Can Do The Same Thing On The Dayside Using Darkest Pixel


2014-09-24


2015-05-15

And The Lightest...


2013-11-14


2015-03-26

Defining and Parallelizing Work (Task framework)

Running Tasks

- Manipulating images takes time
- Generating frames in movies takes time
- Don't want to worry about concurrency or parallelizing tasks
- RabbitMQ and Celery
- Django command framework
- Django ORM
- django_celery

tasks.py in Django App

```
@task()
def insert(filepath, check_for_existing=True):
    info = Info.insert(filepath, check_for_existing)
```

Running Tasks in Django App

```
def handle(self, *args, **options):
    # Make sure our normals are up-to-date
    Normal.update_normals()

    # Go through the list of image files
    for line in fileinput.input(args):
        # Remove spaces and new lines
        line = line.strip()
        tasks.insert.delay(line)
```

Image Gnomes Steps To Success

- Phase 1
  - Collect 6 Million Images
- Phase 2
  - Make Art and Do Science
- Phase 3
  - ???
- Phase 4
  - PROFIT!!

Science Can Be Beautiful, And Art Can Teach

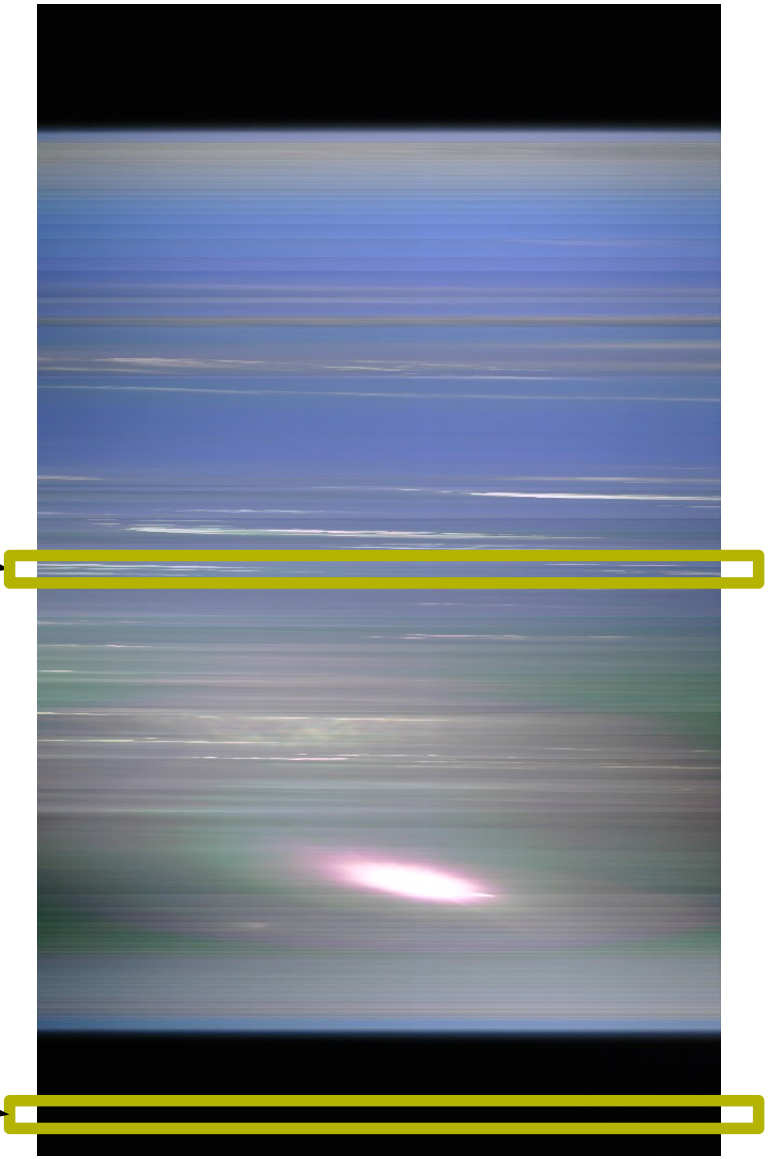
Daystrips:

Visualize a day in a single image

Daystrip Scanner

12:06pm

11:40pm

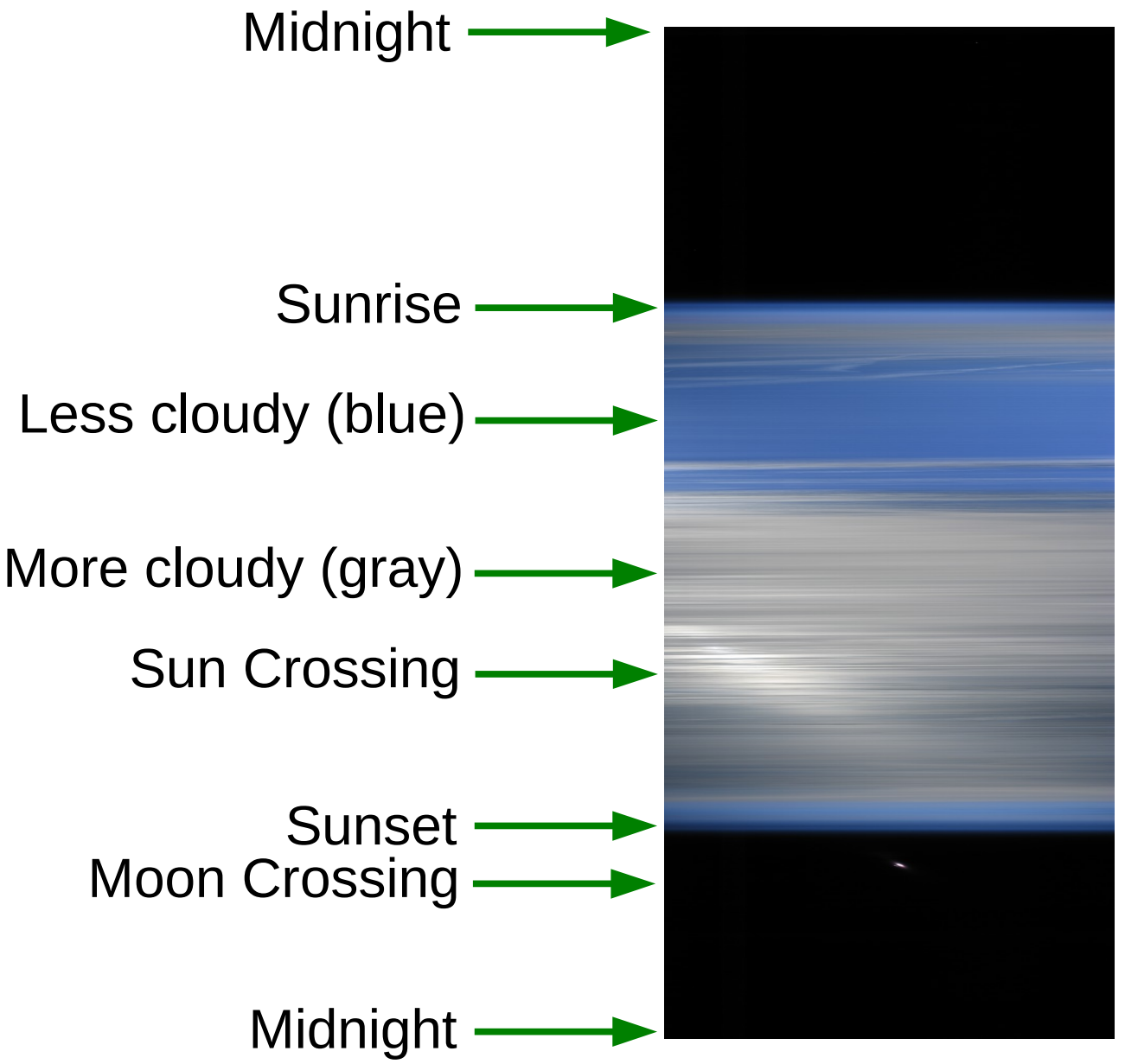


```
@task()
def make_daystrip(dirpath, day):
    # Normalize to midnight
    day_start = datetime(day.year, day.month, day.day, tzinfo=est)
    dayname = day.strftime('%Y-%m-%d')

    day_end = day_start + timedelta(days=1)

    normals = Normal.objects.filter(timestamp__gte=day_start,
                                    timestamp__lt=day_end)

    img = Image.new("RGB", (2048,8640))

    for row, normal in enumerate(normals):
        if normal.picture is None:
            continue

        picture = Image.open(normal.picture.filepath)
        img.paste(picture.crop((0,400,2048,401)),(0,row))

    img.save(os.path.join(dirpath, '{0}.png'.format(dayname)))
```

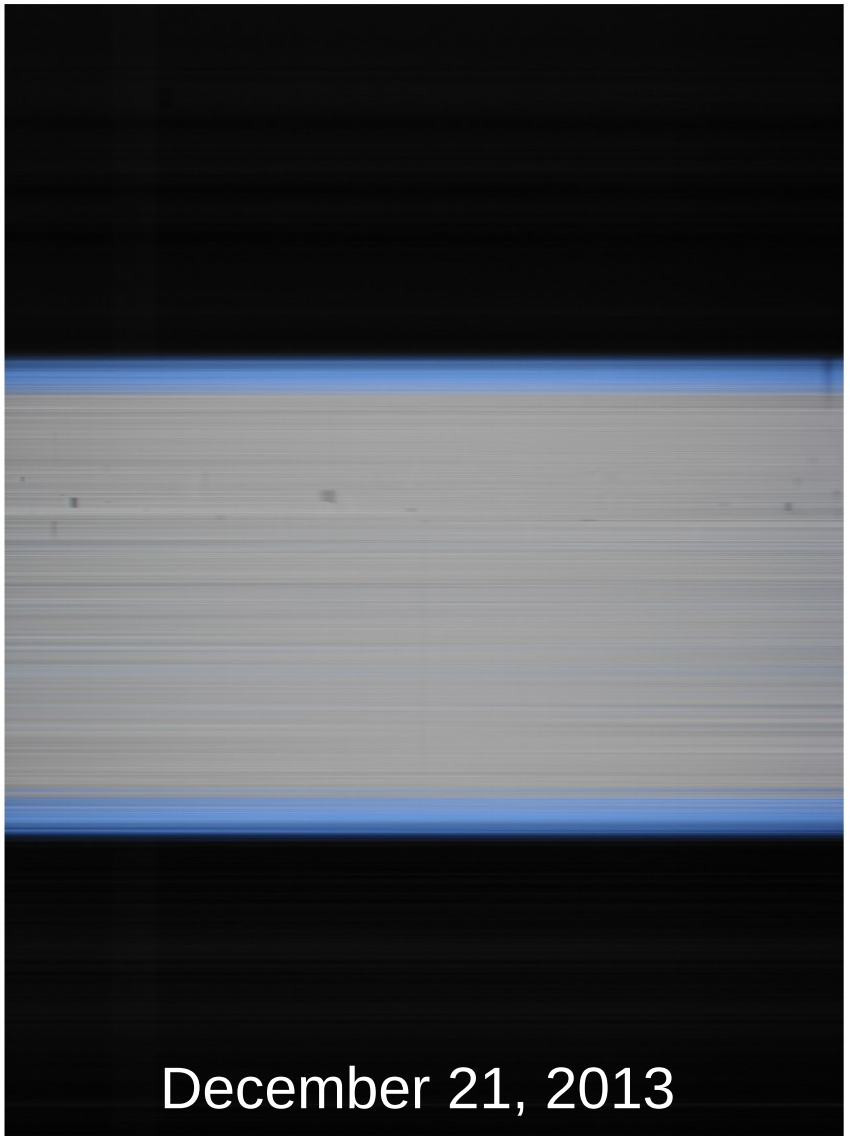
```
@task()
def make_daystrip(dirpath, day):
    # Normalize to midnight
    day_start = datetime(day.year, day.month, day.day, tzinfo=est)
    dayname = day.strftime('%Y-%m-%d')

    day_end = day_start + timedelta(days=1)

    normals = Normal.objects.filter(timestamp__gte=day_start,
                                    timestamp__lt=day_end)

    img = Image.new("RGB", (2048,8640))

    for row, normal in enumerate(normals):
        if normal.picture is None:
            continue

        picture = Image.open(normal.picture.filepath)
        img.paste(picture.crop((0,400,2048,401)),(0,row))

    img.save(os.path.join(dirpath, '{0}.png'.format(dayname)))
```

```
@task()
def make_daystrip(dirpath, day):
    # Normalize to midnight
    day_start = datetime(day.year, day.month, day.day, tzinfo=est)
    dayname = day.strftime('%Y-%m-%d')

    day_end = day_start + timedelta(days=1)

    normals = Normal.objects.filter(timestamp__gte=day_start,
                                    timestamp__lt=day_end)

    img = Image.new("RGB", (2048,8640))

    for row, normal in enumerate(normals):
        if normal.picture is None:
            continue

        picture = Image.open(normal.picture.filepath)
        img.paste(picture.crop((0,400,2048,401)),(0,row))

    img.save(os.path.join(dirpath, '{0}.png'.format(dayname)))
```

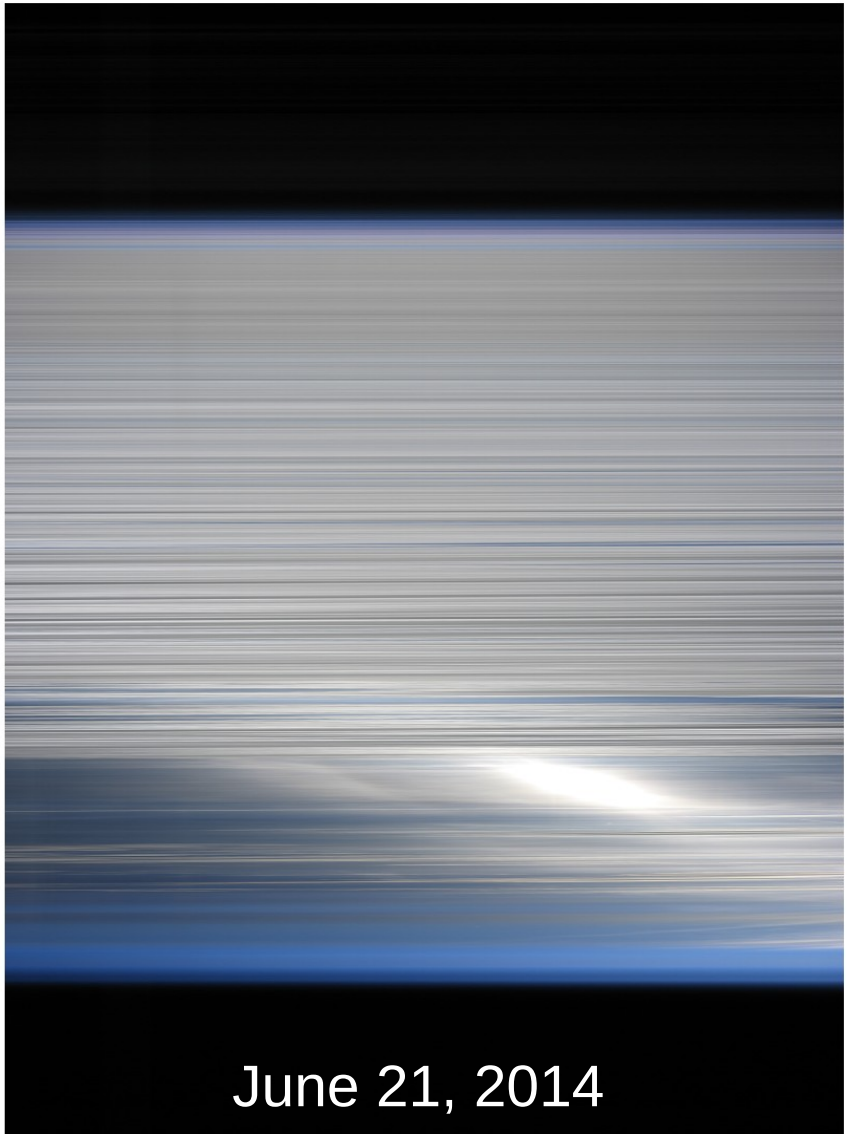
Daystrip Example – March 5, 2014



Shortest and Longest Day

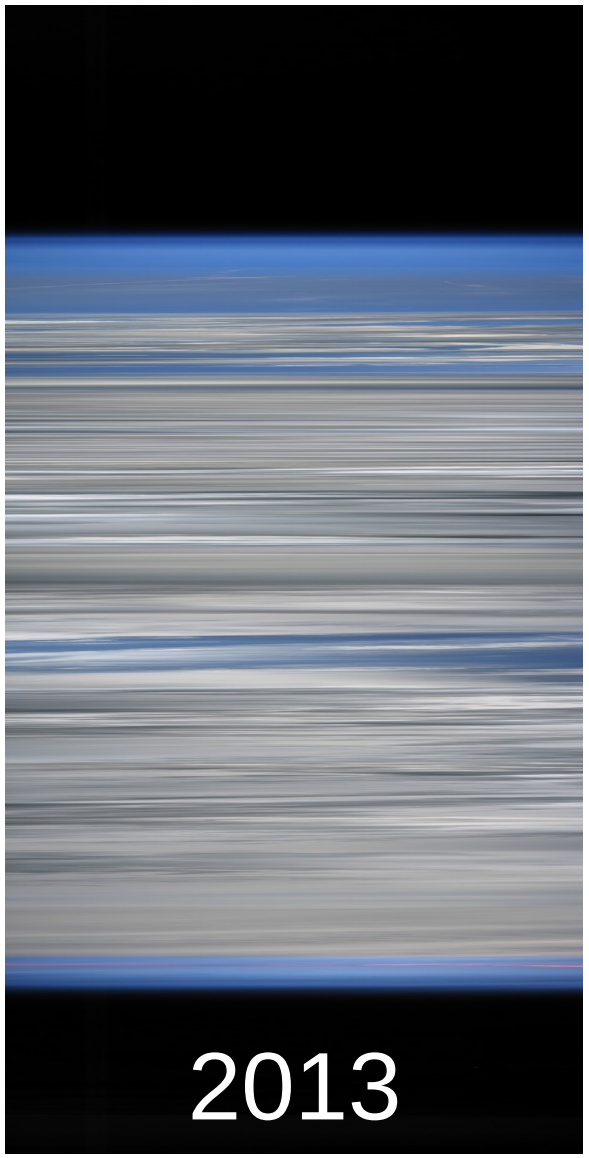


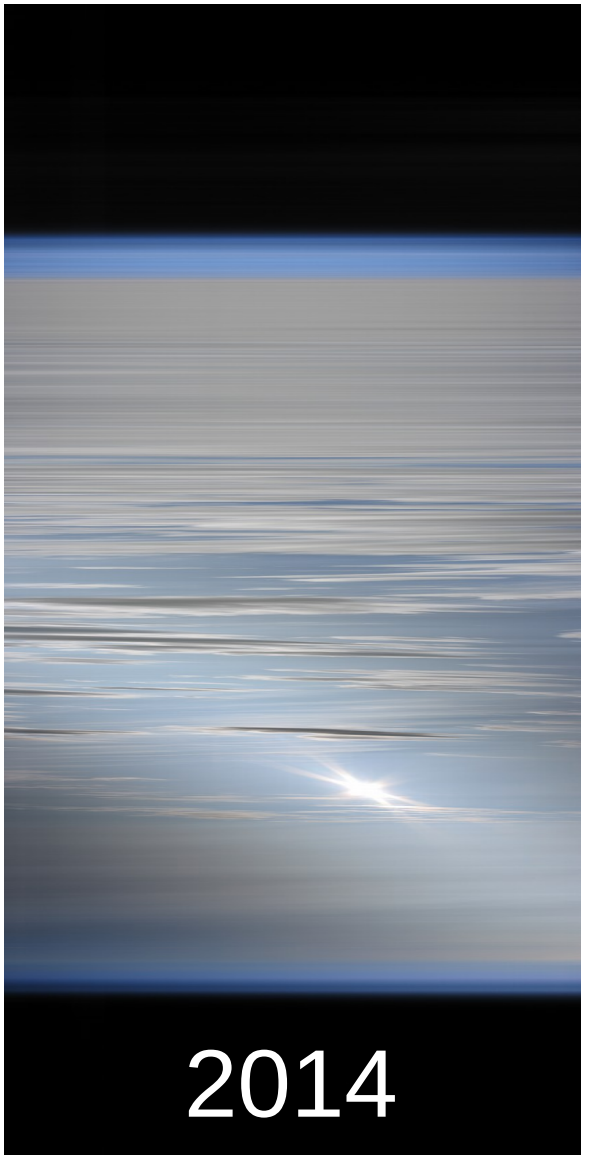
December 21, 2013

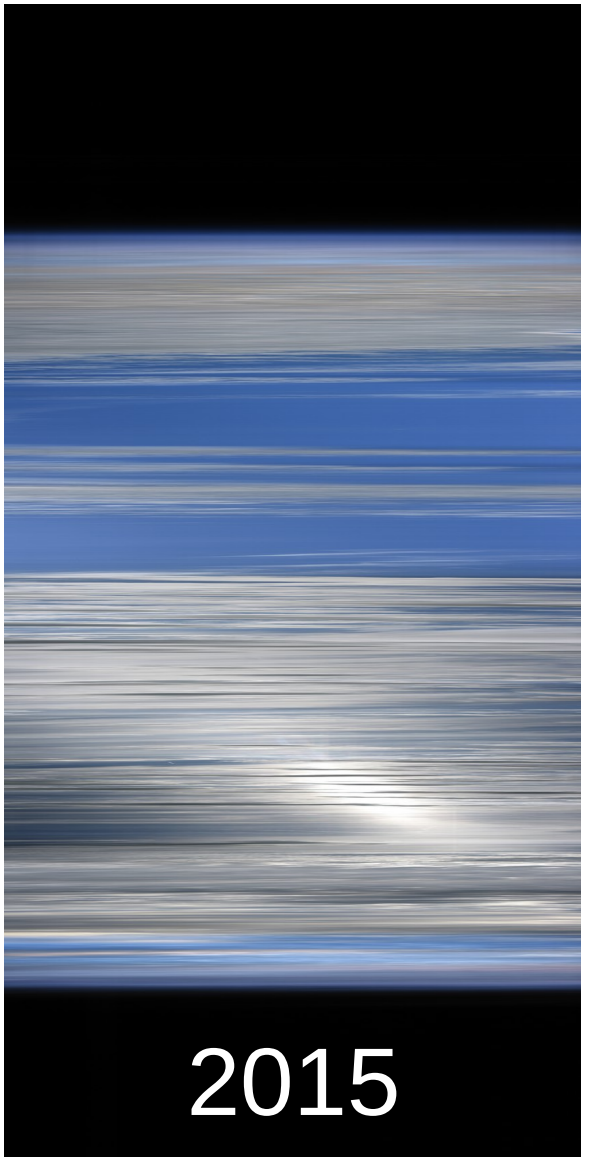


June 21, 2014

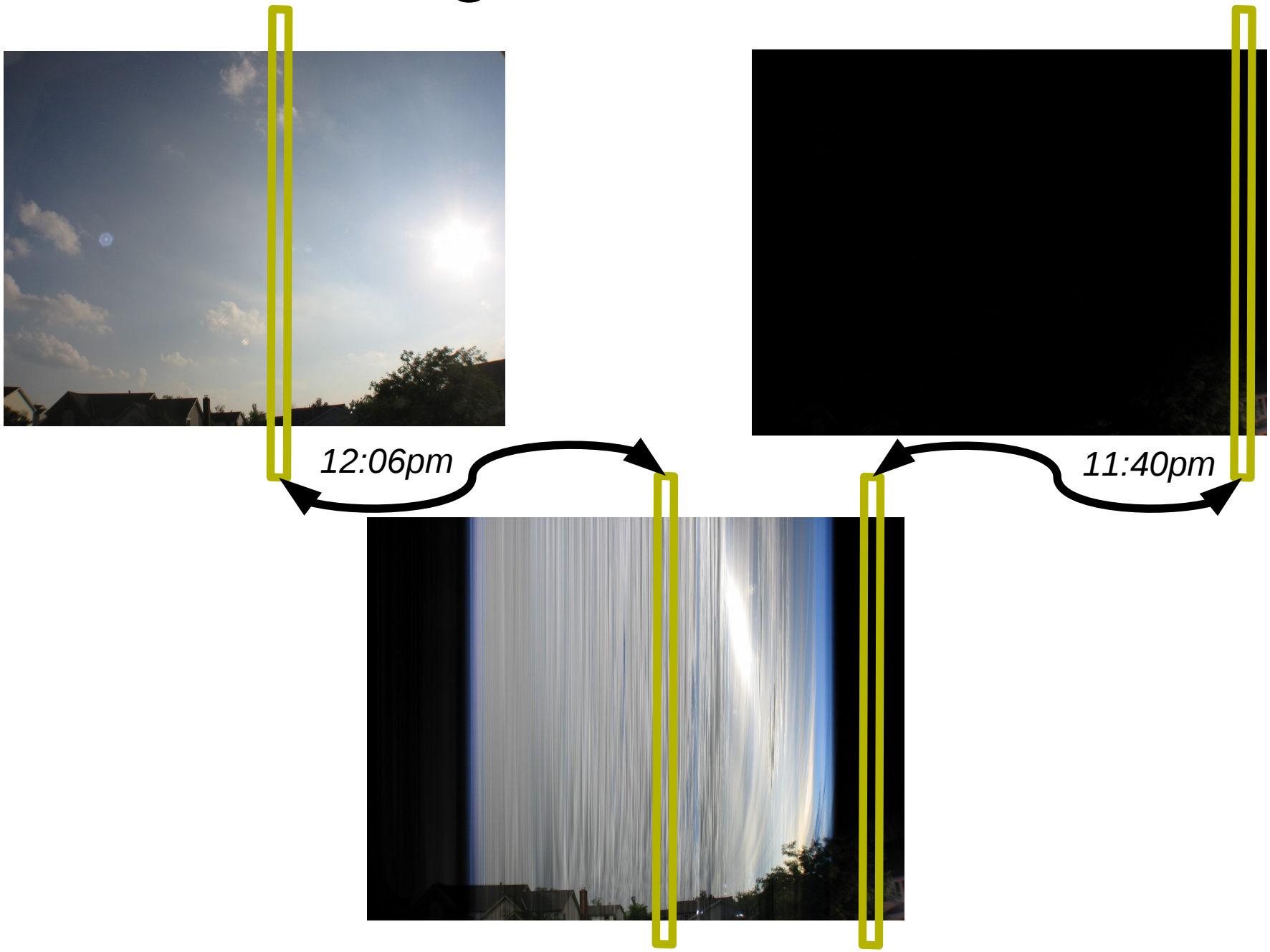
July 20







Scanning to Make a Picture



```
@task()
def make_daystrip_picture_vert(dirpath, day):
    # Normalize to midnight
    day_start = datetime(day.year, day.month, day.day, tzinfo=est)
    dayname = day.strftime('%Y-%m-%d')

    day_end = day_start + timedelta(days=1)

    normals = Normal.objects.filter(timestamp__gte=day_start,
                                    timestamp__lt=day_end)

    img = Image.new("RGB", (2048,1536))

    current_col = None
    for normal_no, normal in enumerate(normals):
        if normal.picture is None:
            continue

        column = int(math.floor(2048 * (normal_no / 8640.0)))

        if column == current_col:
            continue

        current_col = column
        picture = Image.open(normal.picture.filepath)
        img.paste(picture.crop((column,0,column+1,1536)),(column,0))

    img.save(os.path.join(dirpath, '{0}.png'.format(dayname)))
```

Daystrip Movie

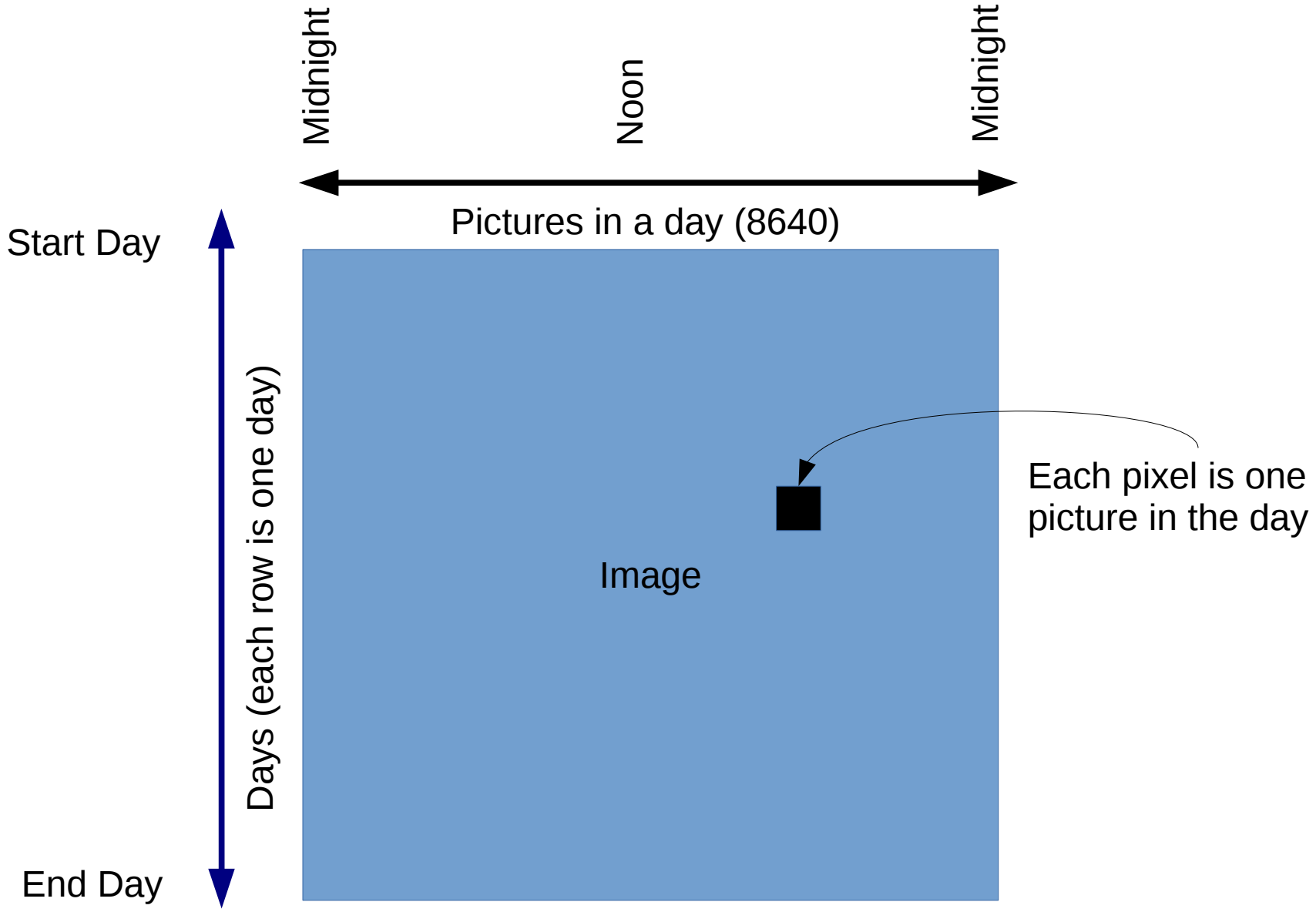
Time is no longer on the Z axis

Time Warp Movie

Let's Use All The Days!

- What if a day were a pixel
- One pixel per image
- One row per day
- 8640 pixels wide (a picture every 10 seconds)
- ~400 pixels tall (one per day)
- We can use median color, max color, etc.

A Visual Representation



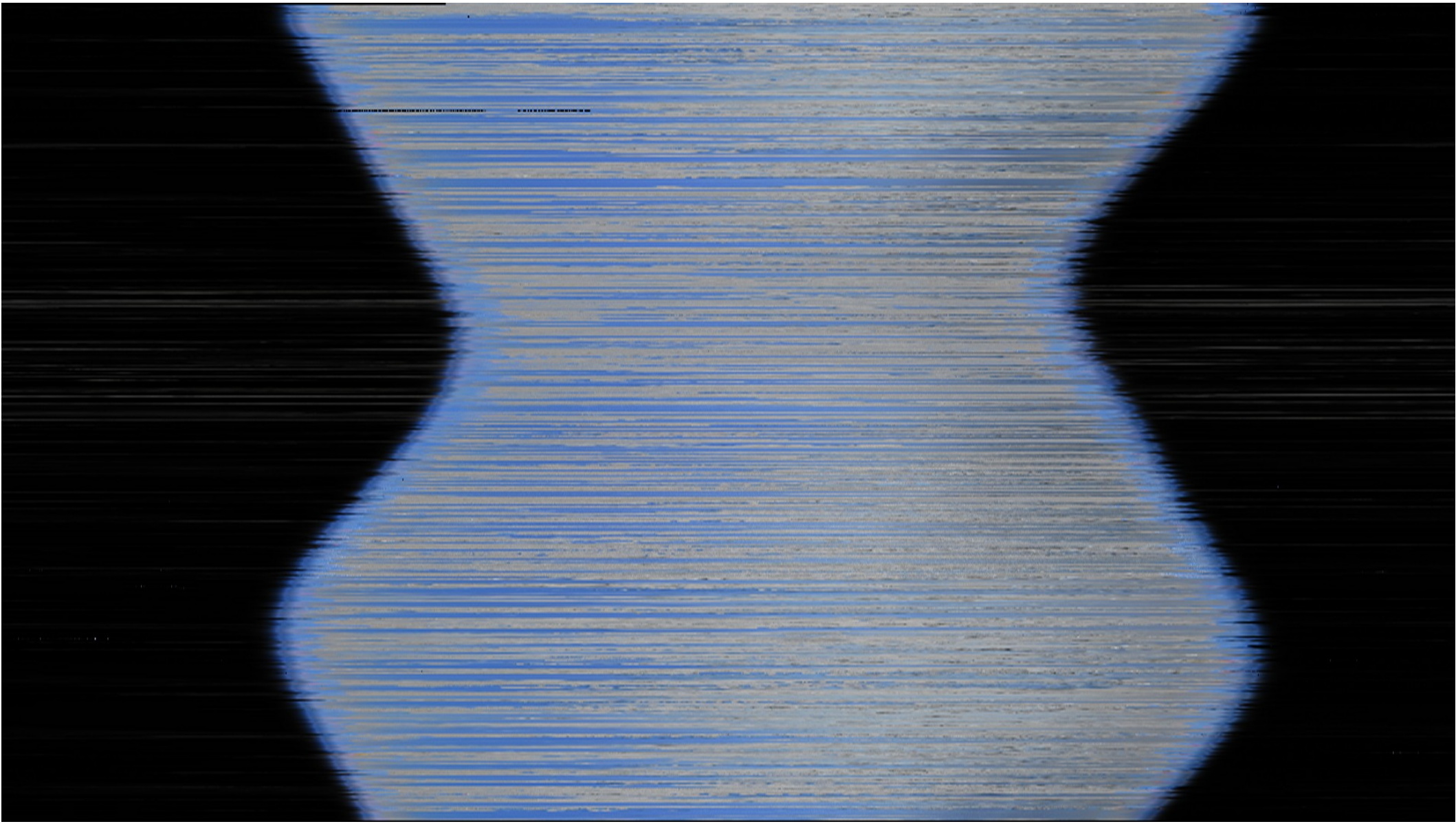
```
#max_color = models.IntegerField(null=True)
im_max = Image.new('RGB', (8640,days))

for normal in normals:
    if normal.picture is None:
        continue

    timestamp = normal.timestamp.astimezone(est)
    datestamp = datetime.combine(timestamp.date(),
                                 time(tzinfo=est))

    y = (timestamp - start_date).days
    x = int(round((timestamp - datestamp).seconds / 10))
    pos = (x,y)

    #max_color = models.IntegerField(null=True)
    color_max = int_to_rgb(normal.picture.max_color)
    im_max.putpixel(pos, color_max)

# All done, save
dirpath = '/var/tlwork/alltime'
im_max.save(os.path.join(dirpath, 'max_raw.png'))

# Scale
im_max.resize((1920,1080),
       Image.BICUBIC).save(os.path.join(dirpath, 'max_hd.png'))
```

```
#max_color = models.IntegerField(null=True)
im_max = Image.new('RGB', (8640,days))

for normal in normals:
    if normal.picture is None:
        continue

    timestamp = normal.timestamp.astimezone(est)
    datestamp = datetime.combine(timestamp.date(),
                                 time(tzinfo=est))

    y = (timestamp - start_date).days
    x = int(round((timestamp - datestamp).seconds / 10))
    pos = (x,y)

    #max_color = models.IntegerField(null=True)
    color_max = int_to_rgb(normal.picture.max_color)
    im_max.putpixel(pos, color_max)

# All done, save
dirpath = '/var/tlwork/alltime'
im_max.save(os.path.join(dirpath, 'max_raw.png'))

# Scale
im_max.resize((1920,1080),
       Image.BICUBIC).save(os.path.join(dirpath, 'max_hd.png'))
```

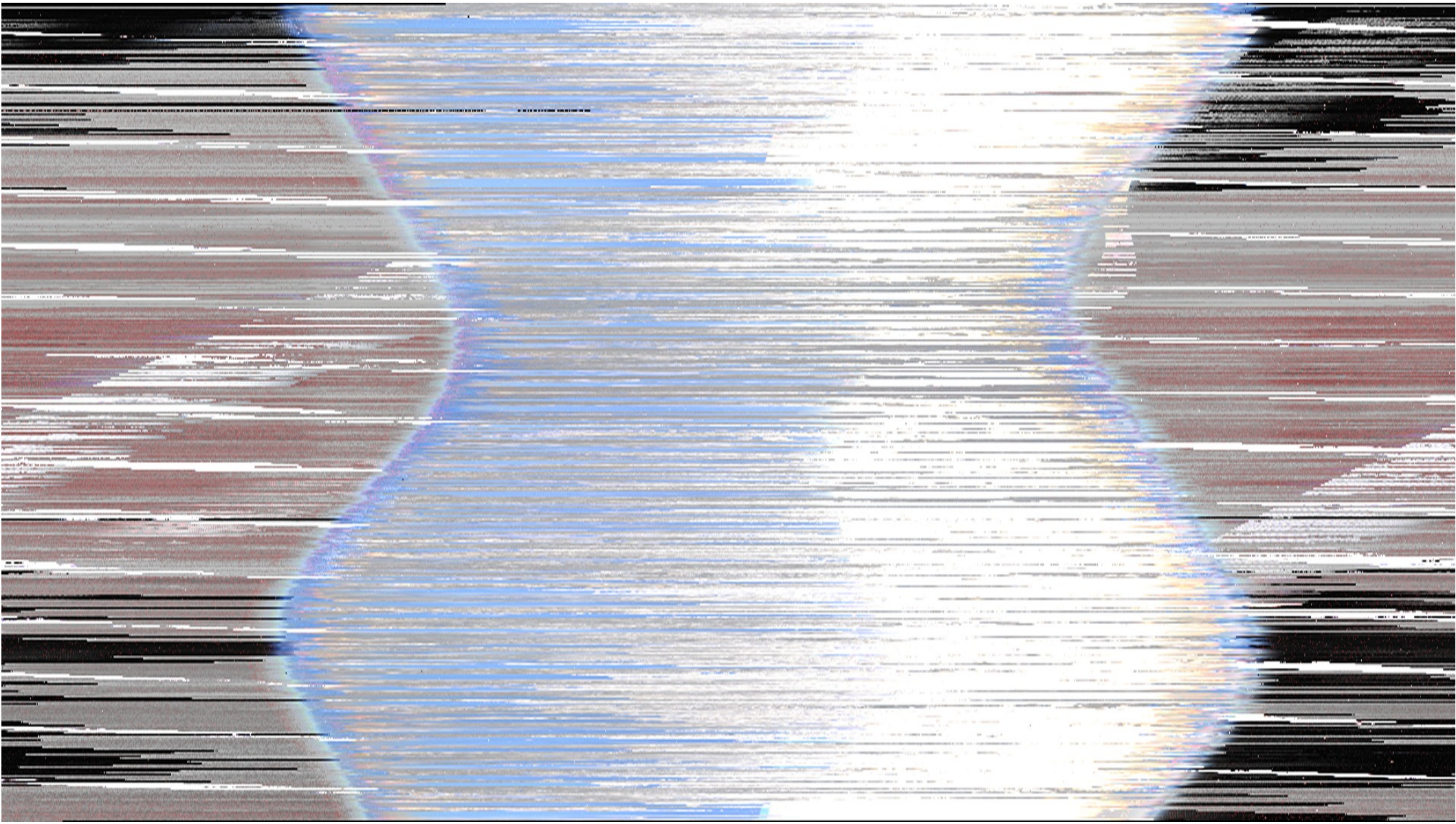
```
#max_color = models.IntegerField(null=True)
im_max = Image.new('RGB', (8640,days))

for normal in normals:
    if normal.picture is None:
        continue

    timestamp = normal.timestamp.astimezone(est)
    datestamp = datetime.combine(timestamp.date(),
                                 time(tzinfo=est))

    y = (timestamp - start_date).days
    x = int(round((timestamp - datestamp).seconds / 10))
    pos = (x,y)

    #max_color = models.IntegerField(null=True)
    color_max = int_to_rgb(normal.picture.max_color)
    im_max.putpixel(pos, color_max)

# All done, save
dirpath = '/var/tlwork/alltime'
im_max.save(os.path.join(dirpath, 'max_raw.png'))

# Scale
im_max.resize((1920,1080),
       Image.BICUBIC).save(os.path.join(dirpath, 'max_hd.png'))
```

```
#max_color = models.IntegerField(null=True)
im_max = Image.new('RGB', (8640,days))

for normal in normals:
    if normal.picture is None:
        continue

    timestamp = normal.timestamp.astimezone(est)
    datestamp = datetime.combine(timestamp.date(),
                                 time(tzinfo=est))

    y = (timestamp - start_date).days
    x = int(round((timestamp - datestamp).seconds / 10))
    pos = (x,y)

    #max_color = models.IntegerField(null=True)
    color_max = int_to_rgb(normal.picture.max_color)
    im_max.putpixel(pos, color_max)

# All done, save
dirpath = '/var/tlwork/alltime'
im_max.save(os.path.join(dirpath, 'max_raw.png'))

# Scale
im_max.resize((1920,1080),
       Image.BICUBIC).save(os.path.join(dirpath, 'max_hd.png'))
```

Using Median Color



Using Max Color



What if we put videos of each day in a mosaic?

We could sun-synchronize the times, so each day would show the same time before sunset.

Sunset Mosaic

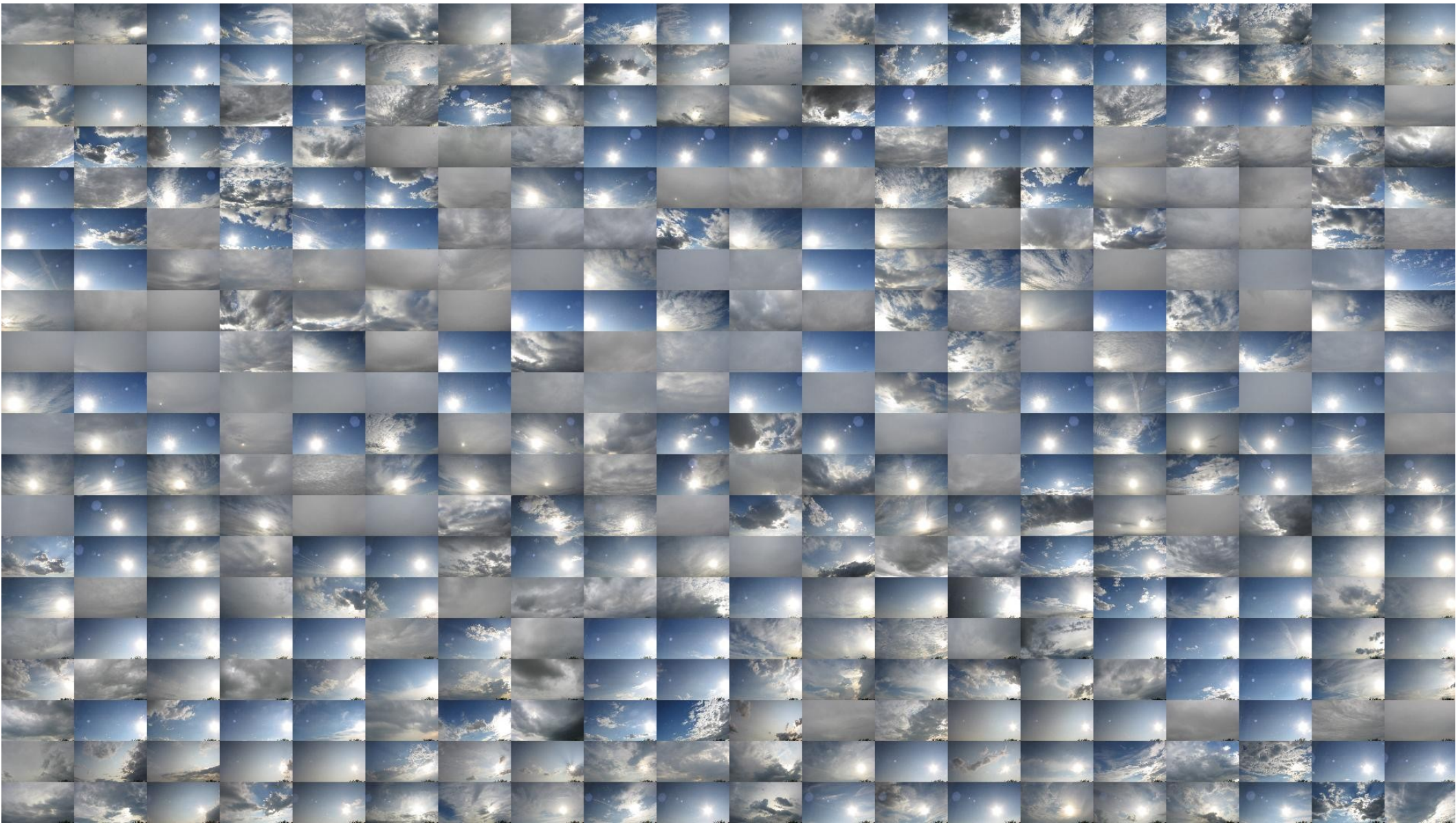
```
    for row_no in range(0,rows):
        row_offset = row_no * scale_height

        for column_no in range(0,columns):
            column_offset = column_no * scale_width
            time = next(times)

            if time.picture is not None:
                img = Image.open(time.picture.filepath)
                imgwidth, imgheight = img.size
                cropbox = (0, start_row, imgwidth, start_row +
                                        int(round(imgwidth * scale_ratio)))
                img = img.crop(cropbox)
                img = img.resize((scale_width, scale_height))
                imgloc = (column_offset, row_offset)
                frameimg.paste(img, imgloc)
```

Sunset Mosaic

```
for row_no in range(0,rows):
    row_offset = row_no * scale_height

    for column_no in range(0,columns):
        column_offset = column_no * scale_width
        time = next(times)

        if time.picture is not None:
            img = Image.open(time.picture.filepath)
            imgwidth, imgheight = img.size
            cropbox = (0, start_row, imgwidth, start_row +
                                    int(round(imgwidth * scale_ratio)))
            img = img.crop(cropbox)
            img = img.resize((scale_width, scale_height))
            imgloc = (column_offset, row_offset)
            frameimg.paste(img, imgloc)
```

Sunset Mosaic

```
for row_no in range(0,rows):
    row_offset = row_no * scale_height

    for column_no in range(0,columns):
        column_offset = column_no * scale_width
        time = next(times)

        if time.picture is not None:
            img = Image.open(time.picture.filepath)
            imgwidth, imgheight = img.size
            cropbox = (0, start_row, imgwidth, start_row +
                                  int(round(imgwidth * scale_ratio)))
            img = img.crop(cropbox)
            img = img.resize((scale_width, scale_height))
            imgloc = (column_offset, row_offset)
            frameimg.paste(img, imgloc)
```



400 Sunsets Video

Image Analysis
(Computer Vision)

PySimpleCV Example

```
# Find a face
scalesize = 1.0
padding = 1.5
scaledimg = img.scale(scalesize)

scalar = 1/scalesize
faces = scaledimg.findHaarFeatures("face.xml",
                                   min_neighbors=17,
                                   min_size=(50,50))

if faces:
    for facenum, face in enumerate(faces):
        scaled_x = face.x * scalar
        scaled_y = face.y * scalar
        scaled_width = face.width() * scalar * padding
        scaled_height = face.height() * scalar * padding

        faceimg = img.crop(face)

        faceimg.save(os.path.join(resultpath, "{0}-{1}.jpg"\
                     .format(image.split('.')[0], facenum)))
```

PySimpleCV Example

```
# Find a face
scalesize = 1.0
padding = 1.5
scaledimg = img.scale(scalesize)

scalar = 1/scalesize
faces = scaledimg.findHaarFeatures("face.xml",
                                   min_neighbors=17,
                                   min_size=(50,50))

if faces:
    for facenum, face in enumerate(faces):
        scaled_x = face.x * scalar
        scaled_y = face.y * scalar
        scaled_width = face.width() * scalar * padding
        scaled_height = face.height() * scalar * padding

        faceimg = img.crop(face)

        faceimg.save(os.path.join(resultpath, "{0}-{1}.jpg"\
                        .format(image.split('.')[0], facenum)))
```

Faces in Clouds

Open Source Makes it Possible!

- Linux
- gphoto2
- bash
- openssh
- Python
- PostgreSQL
- RabbitMQ
- Django
- Celery
- pillow
- pytz
- Pyephem
- pyxif
- OpenCV
- PySimpleCV

With Python, rarely is it the case that the tools lag your imagination

Thank You!

eric@intellovations.com

https://github.com/efloehr/timelapse