My Times :

Parallel Image Seam Computation when nprocs() = 4 on my local MacOS

Stata Image:

Mine: 4.561442 seconds (458.10 k allocations: 158.787 MiB, 3.49% gc time) Regular: 8.152448 seconds (130.98 M allocations: 10.995 GiB, 29.89% gc time)

Yosemite Image:

Mine: 566.174219 seconds (3.72 M allocations: 24.474 , 3.78%) Regular : FOREVER

1200x628 Image:

Mine: 4.912977 seconds (31.58 k allocations: 433.605 MiB, 12.71% gc time) Regular : 31.357681 seconds (452.65 M allocations: 37.977 GiB, 28.80% gc time)

In [11]:
```
@everywhere using Images
@everywhere using Colors
@everywhere using Interact
@everywhere include("imageseams.jl")
```

Parallel Image Seam computation ¶

We are going to split the image into strips by width based on how many workers are available. For example for an image of width 1000 and 10 cores. We will have 10 smaller image strips of width 100 and the the same height as the original image. We will then compute the resulting image by combining these 10 smaller image strips together.

To summarize the parallel image seam computation works as follows

1. Assume we have an image of width and height W,H and on a cluster with P CPU Cores.
2. Split Image into P strips of width W/P and Height H
3. Run the carve seam algorithm from class on each individual strip on each core parallely
4. After all the cores finish computation - fetch all the resulting strips back
5. Combine the strips to reconstruct the full image -

Other things that could have improved performance:

1. Would be great to figure out a way to blend two strips smoothly together
2. Maybe run the carve in process on seperate threads too

Check if the yosemite file already exists on the file system if not download the image

In [23]:
```
addprocs(max(0, Sys.CPU_CORES-nprocs())) #Add all possible processor
@everywhere sleep(0.1)
# if !isfile("1_yosemite_valley_tunnel_view_2010.JPG")
#     run(`wget http://upload.wikimedia.org/wikipedia/commons/e/ec/1_yosemite_valley_tunnel_view_2010.JPG`)
# end

# img = load("1_yosemite_valley_tunnel_view_2010.JPG")

# if !isfile("self-driving-taxi-header-1200x628.jpg")
#     run(`wget http://thingsautos.com/wp-content/uploads/2017/07/self-driving-taxi-header-1200x628.jpg
# `)
# end

# img = load("self-driving-taxi-header-1200x628.jpg")


if !isfile("Wfm_stata_center.jpg")
    run(`wget http://upload.wikimedia.org/wikipedia/commons/2/25/Wfm_stata_center.jpg`)
end

img = load("Wfm_stata_center.jpg")

H = size(img,1)
W = size(img,2)
println("Size of the image - H: $(size(img,1)) px tall and W: $(size(img,2)) px")
```

```
Size of the image - H: 593 px tall and W: 664 px
```

Split this large image into small image strips and push them into the strips array

In [17]:
```
@everywhere strips = Any[]
num_of_cores = nprocs()
width_of_strips = round(Int,W/num_of_cores) #round to nearest integer

for strip_start=1:width_of_strips:W-width_of_strips+1
    strip_end=strip_start+width_of_strips-1
    push!(strips,img[:,strip_start:strip_end])
end

# This is a utility function (you do not need to understand it)
# which overrides Ijulia's image widget so that manipulate displays with the proper widt
h
immutable ImgFrame
    img::ImageMeta
end
ImgFrame(a::Array{<:AbstractRGB}) = ImgFrame(ImageMeta(a))
Base.show(io::IO, m::MIME"text/html", frame::ImgFrame) =
   write(io, """<img src="$(stringmime(MIME("image/png"), frame.im
g))"/>""")
```

In [18]:
```
@time B = @parallel hcat for i=1:nprocs()
    process(strips[i])
end

# @time A = process(img)
```

```
  4.912977 seconds (31.58 k allocations: 433.605 MiB, 12.71% gc time)
```

In [7]:
```
workers_dim = size(B,2)
strip_width = size(B,1)

X = []
for i=1:strip_width
    D = B[i,1]
    for j=2:workers_dim
        D = hcat(D,B[i,j])
    end
    push!(X,D)
end

@manipulate for image_width=1:strip_width
    ImgFrame(X[image_width])
end
```

Out[7]:

In [24]:
```
@time A = process(img)
```

```
10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180 190 200 210 220 230 240
 250 260 270 280 290 300 310 320 330 340 350 360 370 380 390 400 410 420 430 440 450 46
0 470 480 490 500 510 520 530 540 550 560 570 580 590 592 600 610 620 630 640 650 660
8.152448 seconds (130.98 M allocations: 10.995 GiB, 29.89% gc time)
```

Out[24]: