# Video Retargeting with Paralellized Seam-Carving

SPM Course project
2017

## I. Seam carving for video retargeting

Video retargeting is an adapting of video in size and resolution to the variety of electronic devices when this video supposed to be displayed.

One of the effective methods of video retargeting is based on seam carving, an algorithm of context aware image resizing.

The algorithm was introduced 10 yeas ago (S. Avidan and A. Shamir 2007, [1]) and was included in a broad range of applications (Photoshop CS4, GIMP, digiKam, ImageMagick).

It starts with building pixel-wise energy map of the image: to each pixel is assigned a value that depends on it's importance for image perception. Then, to reduce the dimension of the image by one with the minimal damage to it's content, the algorithm searches one-pixel-wide path from top to bottom (or from left to right) that consists of the least important pixels. This operation of seam search and removal is repeated till an image of desired size is obtained.

The algorithm is proven to work well for the broad category of images. There are a lot of methods of importance function enhancement that allow to prevent eventual artifacts in resized image ([2, 3]).

Applied to video retargeting problem, seam carving algorithm imposes additional time coherence constraint on seams searched in every video frame: to be removed without noticeable impact to viewer experience two corresponding seams in adjacent frames should not be too distant. This constraint is usually formalized as an additional term in importance function, that can be viewed as a measure of the difference in energy between the seam found in current frame and the seam found in previous frame and placed in current frame ([4]).

## II. Parallised seam carving

### i. Importance (energy) map

Seam carving is computationally extensive problem (at least linear in the number of image pixels). Solving it in sequential manner does not meet the modern real time video streaming needs. The already reported results on highly parallelized video retargeting on GPU (with speed up 33,5 fps [5]) made the experiments with video processing in parallel highly appealing.

For our student project we tried to structure the video processing process as a graph of parallel executors and to implement it using FastFlow framework [6].

As wanted to address the problem of real time (or streaming mode) video retargeting, video input was broken out in frames (in contrast, for example, to graph-based surface carving in 3 dimensional video block [3]). Then we processed video frames sequentially.

Because of time coherence constraint the frames are non independent, and the seam i in the frame corresponding to time t can be calculated only after i*th* seam in the previous frame (t-1) is found.

For single frame resizing we experimented with the following in-parallel scheme viewed as a three stages pipeline: importance map calculation, seams search, seam removal.

Importance function is applied to every pixel independently and can be evenly distributed between parallel workers.

The baseline implementation of the image energy function *e(i, j)* is a local gradient filter, simply summing the intensity differences between each pixel and it's immediate neighbors. This simple function is proven to work well on a broad range of images.

Human visual perception is sensitive to gradient changes. The most extreme gradient changes occur at edges, so the energy calculation can be enhanced using edge-emphasizing filters. For our project we use simple Sobel[1] filter. Passed over the image both vertically and horizontally, it emphasizes edges and gradient inconsistencies in textures[2].

**ii. Seams search**

Two well known approach for seam search are graph based min-cut/max-flow algorithm and dynamic programming with pixel-by-pixel advancing of the seam, choosing the least important pixel among the adjacent pixels on every step.

Dynamic programming algorithm (see Figure 1) traverses the image from the second row to the last row and compute the cumulative minimum cost M for all possible connected seams for each entry (i, j):

$$M(i, j) = e(i, j) + \min(M(i-1, j-1), M(i-1, j), M(i-1, j+1))$$

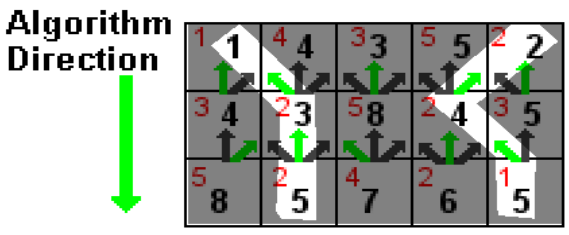



**Figure 1.** Dynamic programming algorithm for seam carving. Each square represents a pixel, with the top-left value in red representing the energy value of that said pixel. The value in black represents the cumulative sum of energies leading up to and including that pixel.

We stick to this dynamic programming method and implement it parallel version.

1 http://docs.opencv.org/3.2.0/d2/d2c/tutorial_sobel_derivatives.html

2 As it's known that the Sobel filter is vulnerable to noise, to improve the quality of image processing a derivative of Gaussian filter can be tried, first smoothing the image and then emphasizing edges. The derivative of Gaussian filter is more resistant to noise and it emphasizes edges well.

We start candidate seams from every pixel of the top row. Till the end of the search process we keep all seams in the memory, encoding them as a sequences of image column indices, and we update their energy values and their length at every seam-growing step.

What we exactly do is: we loop through the image rows and for every pixel in it we calculate it's cumulative value as if this pixel belongs to the seam: we sum energy of current pixel with minimal cumulative value of his upper row neighbors and then we include position of this best neighbor in the seam path. That is, when processing the pixel in the row we update the candidate seam total energy, it's length, and we update the seam trace advancing it by one position.

This calculation can be easily distributed among «column» workers for each independent «row» iteration. It can be easily implemented using FastFlow *ParallelFor* Pattern.

Arrived to the bottom of the image it is necessary to choose the minimal energy seam for removal among all the candidates. Once removed, the chain of operations - of energy map calculation, seam search and seam removal - should be repeated till the image size is reduced by desired amount of pixels, one per step.

C.-K. Chiang et al. [3] propose more effective approach to the problem, when the image is searched for several seams per step. Several resulting seams means that they are seams ready for removal from image and, thus, they are minimal and they do not intersect.

By algorithm construction, as described above, we arrive at the bottom of the image with multiple intersecting seams, those number is equal to image colums number. To obtain more than one isolated seams it's necessary to resolve conflicts between two or thee seams choosong the same image pixel to take when growing from top to bottom edge of the image.

C.-K. Chiang et al. ideas are:

1. Once two (or three) seams are intersected at c*th* column of r*th* row, compare their energies and continue with advancing the seam with smaller energy value. In case of the tie, break it calculating a length of each seams and preferring shortest one.

2. Once you have two almost isolated seams that intersect at the short distance from the bottom (the threshold value to measure, if it is the case, can be introduced to the system), split them choosing the second (and, possibly, third) seam alternative position following reasonable heuristic rule.

We implemented the both types of conflict resolution (with no threshold) to find as more seams as possible in a single step. Instead of searching for one minimal seam we tried to search for several isolated seams at once. And, depending on the content of an image (frame), we expected to find all seams to be removed in a single search step.

We start, as described before, with number of seams equal to image columns number. Then we loop through images rows by row and we assign a parallel worker to every column position in that row. Each worker controls all the candidate seams leaded to this column position by seam search algorithm and, if they are more than one, keeps only one assigned to this position and discards other candidates.

All workers together contribute to the knowledge of the seams survived at the moment of current row processing, so that next row iteration of seam advancing can consider only these seams. Number of seams under consideration drastically decreases with each iteration, and after some number of row processed and majority of candidate seams dropped, it's reasonable to switch from parallel to sequential calculation for remaining seams.

In the end, if more than needed seams have been found, they should by sorted by energy value to drop the seams with greater energy. If the number of seams is less than needed to reduce image dimension the process is repeated.

**iii. Seams removal**

Seams removal as well as seam search (parallelized part) is implemented using ParallelFor pattern. For the seam removal stage all image rows are distributed between the workers. Each worker sorts positions in the row belonging to seams and then copy pixels from row of input image to row of output image increasing offset of input pixel position by one at each seam.

**iv. Coherence term of the energy map**

Starting with the second frame coherence part should be included into importance map forcing correlation in seams positions of the neighboring frames. The coherence term is the difference in energy map of the image from which the minimal seam is removed and the energy map from which removed the corresponding minimal seam found for the the previous frame. It's equal to zero when the corresponding seams of the neighboring frames are the same. To calculate it value recursive formula reported in might be used[4]. This calculation can be again parallelized on the image row level.

In the following section we show that our video retargeting implementation produces acceptable results when reducing image dimension up to approximately 10% of it's original dimension and should be improved when addressing the problem of crucial reduction in size.

We were not interested much in achieving high quality of resulting video (this goal can be addressed with careful energy function choice), we were rather concentrated on the performance issues, trying to achieve noticeable speed up of video processing.

## II. Experiments settings

### i. Input data.

In our experiments we used two video sequences of differing image structure: featured film fragment (Cenerentola of Jean-Pierre Ponnelle, 1981) and cartoon piece with resolutions 640 x 480 (videoplayback.mp4) and 720 x 528 (Megamind.mp4) correspondingly[3].

### ii. Hardware

All the tests were performed on two architectures

3 We also used a several images of different resolutions (487x487=237169,1000x633=633000, 960x960=921600, 2048x1536=3145728) to test our results obtained for video frames

1. On a PC (*Home*) with the following specifications: Intel i7-4790 processor CPU with four cores at 3.6 GHz, 16 GB DDR3 RAM. In the experiments, eight CPU threads are used as the PC has four cores and is able to support two threads per core.

2. On the workstation (*Unipi*) with following characteristics: AMD 6176 0.8 GHz, 2 Sockets, 4 NUMA nodes, 24 CPU cores total, 16 GB DDR3 RAM.

## II. Experiments results

To monitor our program performance we have run several experiments to approximately estimate it's speedup, scalability and efficiency on sample videos.

In our experiments we are faced with a problem of calculating coherence term of image importance map in situation when we do not know how many seams will be found on the each step of minimal seams search, and to what seams in the previous frame they have to correspond. The coherence term described above for the single seam search can not be directly (without revising) applied to the multiple seams search. For the moment we removed the coherence problem from consideration.

The way of searching several seams at once is proved to be quite effective. For our sample video it removes about 20 seams at once. This number depends on frame content and can vary in the broad interval. Figure 1a.-d. demonstrates this fact.

But the process of searching and removing 20-60 seams in each frame of standard video resolution, when parallelized, does not bring noticeable speedup, as the workers are not involved in any kind of extensive calculations that can make negligible the parallelization overhead. On the contrary, we ended up with the negative speed up with seams search calculated in parallel.

Indeed, seams search of our implementation remains rather sequential: the performance of algorithm is crucially improved by searching for a dozens of seams simultaneously and discarding conflicting seams instead of searching a single seam between two energy map recalculations.

However, this simple procedure of resolving seams conflict allow to obtain, on average, only 20 - 50 isolated seams per step (depending on frame content) . And this reduction of the number of valid seam candidate out of n candidates (where n is a number of row pixels) happens on the very first steps on seam advancing.

So, in the end we were interested only in parallel calculation of the energy map.

So, after obtaining negative speedup we have modified the program **to calculate in parallel only energy map**.

Because of the fact that the single frame processing has sequential nature, expressed in ordered steps, the performance from parallel calculation is limited by the slowest sequential unit in the chain, that is by the seams search/seam removal. And so we do not expected a great speedup as a result of parallelized calculation of energy map.

Table 1 show slight speed-up for our video processing on 5% frame size reduction, and no speed-up on 25% reduction, when several hundred seams searched through 10 – 20

iterations, including energy map recalculation. In the later case energy map is calculated for progressively reduced image, so trace it's impact to the overall calculation time become a harder task.

To make the impact of parallelizing of energy map calculation clear, we've run additional experiments on single images of different resolutions. (see one example on Figure 2) The gradient calculation is the convolution problem, it's complexity is **O(mn)**, where m and n are image vertical and horizontal dimensions, so we expected a linear speed-up, proportional to the number of threads involved in calculation.

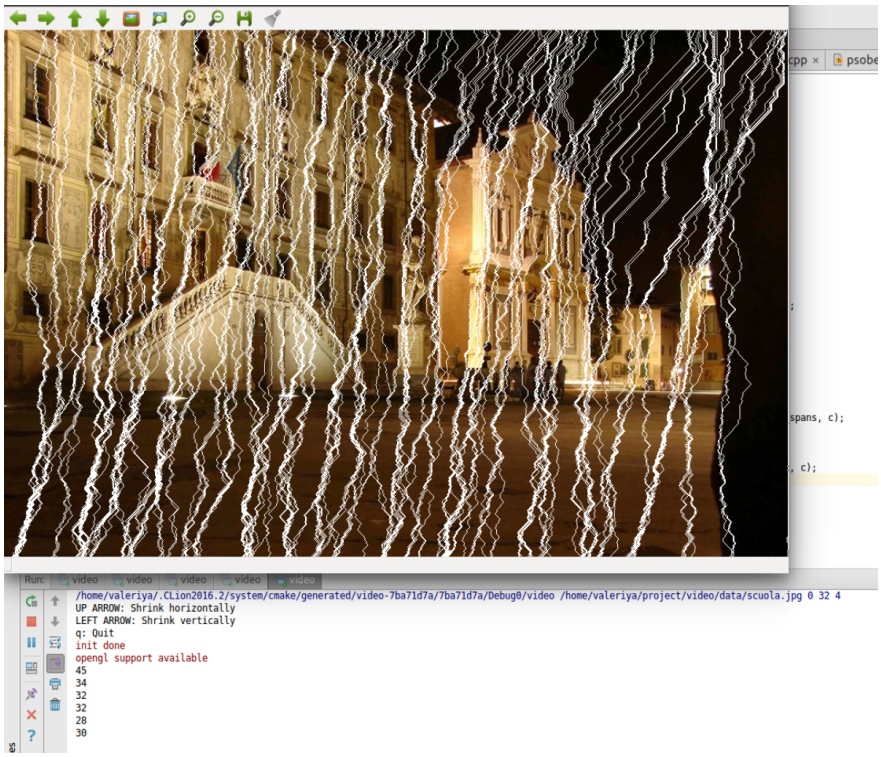



**Figure 2.a**

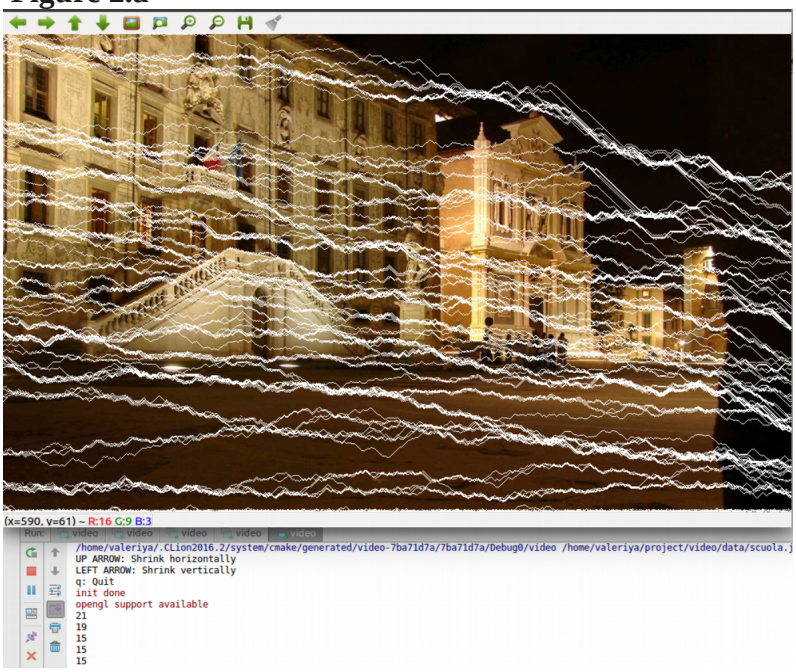
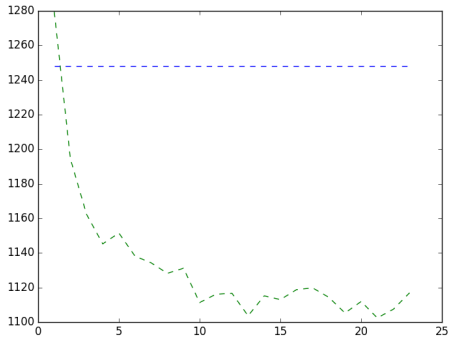
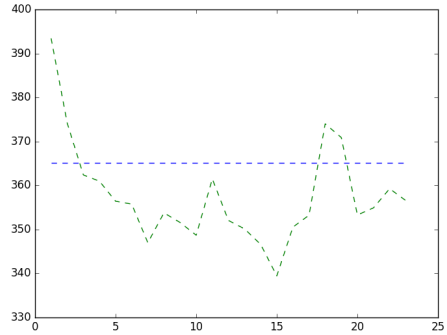



**Figure 2.b**

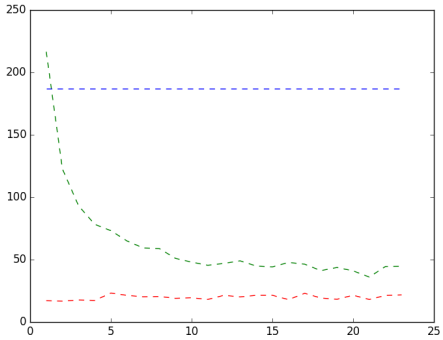
Multiple seam searchs. a. Iteration with 45, 34. 32... vertical seams found. b. Iterations with 21, 19, 15... horizontal seams found.



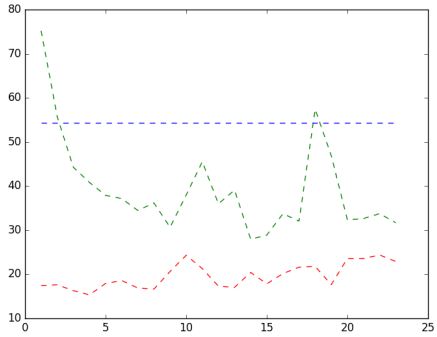

a) 25% size reduction of large image, ms



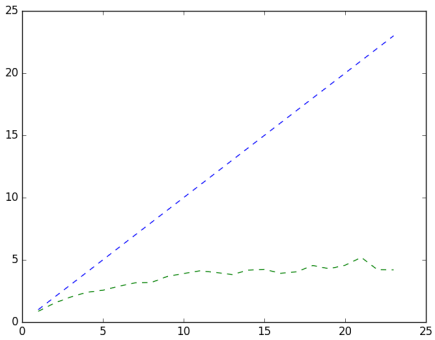

b) 25% size reduction of smaller image, ms



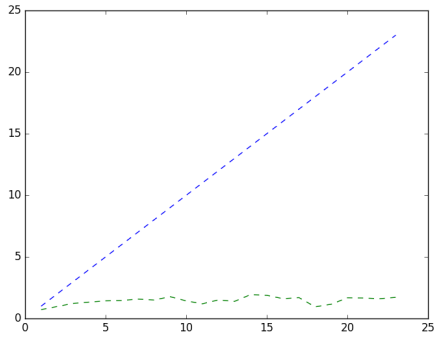

c) Energy map calculation for large image, ms




d) Energy map calculation for smaller image, ms




e) Energy map calculation **speed-up** for large image




f) Energy map calculation **speed-up** for smaller image

**Figure 3.** a. and b. **:** 25% image size reduction time (ms), c. and d. : energy map calculation time (ms), e. and f.: energy map calculation speed-up as a function of the parallel workers number for two images. Left: big image of 2048 rows and 1536 columns, right: smaller image of 960 rows and 960 columns. Horizontal (a),b),c), d)) line corresponds to sequential execution of algorithm. Blue line on e) and f) is an expected speed-up. Red line on c) and d) is the ParallerFor object creation impact to energy calculation.

Our results show speed-up values different from these expectations. (Figure 3) Partially this difference can be attributed to parallelization overhead: for example, 4 to 20 ms is needed for FastFlow ParallelFor object creation. This overhead seams to grow with thread number and makes a negative impact into efficiency (Figure 4).

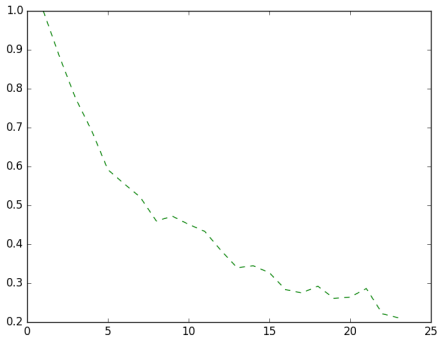



**Figure 4**. Efficiency decay with number of workers. Energy map calculation efficiency for the large image of  of 2048 rows and 1536 columns as a function of parallel workers number,

We can see that energy map calculation in parallel is very efficient for **large images** – order of magnitude large than the frames of video frames we experimented with.

| nw | Home PC<br>5 % size reduction | Home PC<br>25 % size reduction | Unipi WS<br>5% size reduction | Unipi WS<br>25 % size reduction |
|---|---|---|---|---|
| 1 | 96.78 | 1322.16 | 323.56 | 4113.70 |
| 2 | 85.16 | 1694.09 | 300.02 | 5280.49 |
| 3 | 82.47 | 1664.18 | 290.20 | 5084.21 |
| 4 | 82.36 | 1641.10 | 287.67 | 5108.09 |
| 5 | 85.14 | 1756.58 | 280.36 | 4964.30 |
| 6 | 84.68 | 1688.35 | 281.18 | 5068.04 |
| 7 | 85.12 | 1603.58 | 273.73 | 4980.45 |
| 8 | | | 282.90 | 5041.31 |
| 9 | | | 276.33 | |
| 10 | | | 279.83 | |
| 11 | | | 284.14 | |
| 12 | | | 276.93 | |
| 13 | | | 276.40 | |
| 14 | | | 277.39 | |
| 15 | | | 269.28 | |
| 16 | | | 277.60 | |
| 17 | | | 276.44 | |
| 18 | | | 289.30 | |
| 19 | | | 279.52 | |
| 20 | | | 282.35 | |
| 21 | | | 284.32 | |
| 22 | | | 283.22 | |
| seq | 80.58 | 666.58 | 242.78 | 2117.49 |

**Table 1**. Average time (ms) for 5% and 25% frame width reduction measured on Home PC and Unipi Workstation with Cenerentola videosequence of 100 frames as a function of paraller workers number. Last line: time of sequential execution.

To conclude, our experiments have shown some limits that prevents us from achieving noticeable speedup, due to high parallelization degree. The seams search we adopted show itself inherently sequential when only the energy map calculation part can be efficiently parallelized for the high quality videos.

It’s important also to note, that seam carving rises the problem of quality and performance trade-off. Sophisticated algorithms for importance map calculation may have a negative impact on performance (in our case this impact related with coherence term calculation). Alternating horizontal and vertical seam removal can improve quality but it’s more tricky to implement in parallel. Seams splitting, needed to increase the performance of seams search, should be implemented with careful choice of splitting heuristics to avoid degradation of the visual quality of retargeted video.

## IV. User manual

Requirements

To run the program installation of OpenCV version 3.2 and video codecs are required [4].

Installation

1. Unpack the archive or clone git repository:

- $ git clone https://github.com/vslovik/video.git
- $ cd video/parallel

2. Install FastFlow:

- $ svn co https://svn.code.sf.net/p/mc-fastflow/code/ fastflow
- $ svn update

3. Correct path to FastFlow in Makefile if needed

Compillation

- $ cd video/parallel
- $ make clean
- $ make

Usage

- $ ./video path/to/videofile <nv> <nh> <nw>

nv — number of vertical seams to remove

nh — number of horisontal seams to remove

nw — number of workers

4 For automated provisioning with Ansible see: https://github.com/vslovik/videobox

Usage examples

- $ ./video ../data/videoplayback.mp4 0 100 10
- $ ./video ../data/Megamind.mp4 0 100 10

Two sample video files are available in data folder of code repository.

## V. References


[1] S. Avidan and A. Shamir. Seam carving for content-aware image resizing. ACM Transactions on
Graphics, SIGGRAPH 2007, 26(3), 2007.

[2] J. Kiess, S. Kopf, B. Guthier, and W. Effelsberg. Seam carving with improved edge preservation. In Proceedings of IS&T/SPIE Electronic Imaging (EI) on Multimedia on Mobile Devices, volume 7542, pages 75420G:01 – 75420G:11, January 2010.

[3] C.-K. Chiang, S.-F. Wang, Y.-L. Chen, and S.-H. Lai. Fast jnd-based video carving with gpu acceleration for real-time video retargeting. IEEE Trans. Cir. and Sys. for Video Technol., 19(11):1588–1597, 2009.

[4] M. Grundmann, V. Kwatra, M. Han, I. Essa. Discontinuous Seam-Carving for Video Retargeting. Source: Computer Vision and Pattern Recognition (CVPR), June 2010 , pp. 569 – 576.

[5] J. Kiess, D. Gritzner, B. Guthier, S. Kopf, W. Effelsberg. GPU Video Retargeting with Parallelized SeamCrop in MMSys '14 Proceedings of the 5th ACM Multimedia Systems Conference Pages 139-147, Singapore, Singapore — March 19, 2014

[6] M. Aldinucci, M. Torquati, M. Drocco, G. P. Pezzi, C. Spampinato . An Overview of FastFlow: Combining Pattern-Level Abstraction and Efficiency in GPGPUs . GPU Technology Conference (GTC 2014), 2014