Introduction to Computer Vision

Final Project

Moshe Amini – 204000103

Roie Danino – 316440296

24 February 2018

Stereoscopic Panorama from 360 Video



Figure 1 Stereo Panorama



Figure 2 Right Eye



Figure 3 Left Eye

**Abstract**

In this project, we created stereoscopic panoramas from 360 videos using one camera instead of 2 images, using multiple techniques and algorithms such as SIFT for feature detection and extraction, RANSAC for finding the homography, fixing frames curves and rotation changings (that can be cause as a result of shaky hands while filming the video) and moving objects such as tree leafs and animals, we created one panorama for the right eye and another one for the left eye, combining them created a red – cyan 3D images (anaglyph).



Figure 4 Full Anaglyph (Try with red-cyan 3D Glasses)

# stereo%20Mountain1%20.jpgZoom In (Figure 4 vertical cut to enable larger and more convenient perspective)



**Introduction and Background**

Our project is inspired by and based on the papers:

“OmniStereo: Panoramic Stereo Imaging” by Shmuel Peleg, Moshe Ben-Ezra and Yael Pritch, and “Megastereo: Constructing High-Resolution Stereo Panoramas “, we implemented our own version of Stereoscopic panorama images from video, but we used one – camera instead of 2 orthogonal cameras, we implemented the 3D red-cyan anaglyph image combining the left eye image and the right image, but first let’s explain some basic terms and sub examples of the possible applications of our project:

A **panorama** is any wide-angle view or representation of a physical space, whether in painting, drawing, photography, film, seismic images or a three-dimensional model, panoramic pictures usually created with a wide angles camera lens or combining few pictures one next to the other.



Old Stereoscope

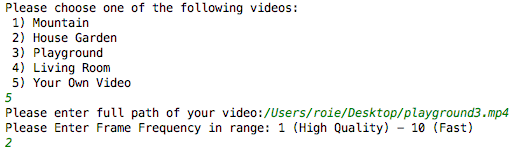
**Stereoscopy view** is a technique for creating or enhancing the illusion of depth in an image by means of stereopsis for binocular vision, creating a 3D feel image from 2 photos of the same scene from different angles. The first stereoscope was invented in 1838, and gave this particular effect, today we got the VR glasses and many smartphones with dual lens that creates the same effect.

**Anaglyph 3D** is the name given to the stereoscopic 3D effect achieved by means of encoding each eye's image using filters of different (usually chromatically opposite) colors, typically red and cyan. Anaglyph 3D images contain two differently filtered colored images, one for each eye. When viewed through the "color-coded" "anaglyph glasses", each of the two images reaches the eye it's intended for, revealing an integrated stereoscopic image. The visual cortex of the brain fuses this into the perception of a three-dimensional scene or composition.

**The Algorithm**

This project is basically taking a single camera video and creating various panoramas:

right eye panorama, left eye panorama, right and left panorama mixed alternately and an anaglyph panorama – red-cyan 3D panorama.

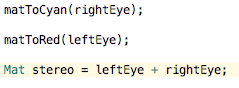
****So the first step is going to be creating an **Interactive Menu**, so the user will be able to choose what video would he like to insert as an input, of course he will need to make sure that the video paths are correct before starting the program, he can either choose one of our videos or insert one of his own by inserting the full path to his video, after choosing a video the user need to insert his desired frame frequency that eventually effect the quality of the result panoramas, when 1 means that every frame will be considerd at the creation of the panorama and 10 means every 10th frame will be considerd.

After the user chose a video we would like to **load and show the input video,** frame by frame, showing every frame and saving in a vector every Fth frame where F is the Frequency the user chose , after saving those frames in a vector we will **fix the frames size** if they are too big to fit the screen.

For fitting all the frames one next to each other, we need to **find the homography** between every following pair of frames, to do so, we need to detect and extract the features from both pictures of every pair using **SIFT** which is great for the job, then we need to match the features in both pictures, we will use the BF Matcher for that purpose which will give us a vector of matches, using this vector and the vectors of features in both images we will create the homography with the RANSAC algorithm that overcomes differences of size rotation, brightness and much more, and we save all of those homographies in a vector too.

Then, we can finally **create, show and save the panoramas**, for creating the panorama we need to decide what width each frame will be responsible for, the right eye panorma will be offseted to the right by that width size which will also be the “fake” **despairity**, so the left eye panorama will be offseted by twice that width, the mixed panorama is created by merging the slices from the right panorama and and the slices from the left one alternately, then, we show (using opencv imshow) and write (using opencv imwrite) those panoramas to a file with the name of the video and the frequency concatenated to it.

For the anaglyph panorama we take the left image and we set the blue and green of each of its pixels to 0, that will make the left eye image red, we do the opposite to the right eye image, we set each of the right eye image pixel red value to 0 so the right eye will be cyan, then, we add those together and create the combined 3D anaglyph panorama, that we will see as 3 dimentional photo with red-cyan 3D glasses.

****

**Results**



Figure 6 Zooming In

Figure 6 Full Panorama



Figure 9 Anaglyph

Figure 8 Left Eye

Figure 8 Right Eye

Figure 7 Left Eye

Figure 7 Right Eye

**Results**

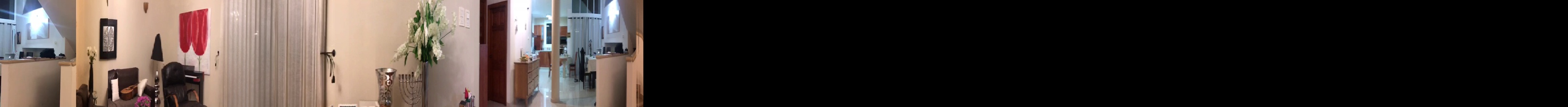


Figure 13 Anaglyph

Figure 11 Right Eye

Figure 12 Left Eye

Figure 10 Inside House Panorama

**Results**





Figure 14 Right Eye



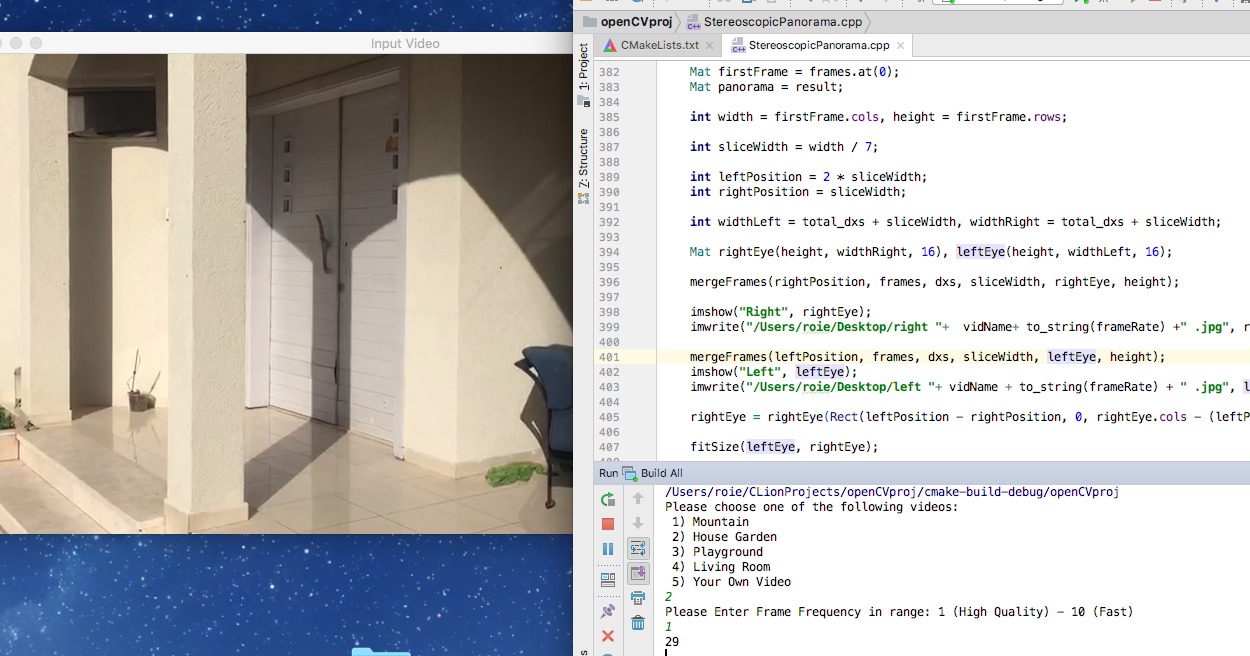
Figure 15 Left Eye



Figure 16 Anaglyph

**Summary and Conclusion**

In conclusion, the creation of stereoscopic panorama is an amazing application of the combination of many subjects in computer vision, such as feature detection, feature extraction, feature matching, finding homographies and merging sub-images, this is wonderful to see how we can achieve the effect of dual cameras images with one camera videos, this task was not simple but very interesting to explore the results with different parameters values – fast videos vs slow videos, high frame frequency vs low, different despairity values and detection methods.

****

**References**

<https://en.wikipedia.org/wiki/Feature_detection_(computer_vision)>

https://en.wikipedia.org/wiki/Homography

https://yafangshih.github.io/stereo-pano/

<https://moodle.afeka.ac.il/pluginfile.php/308346/mod_assign/intro/peleg%20et%20al%202001%20-%20Omnistereo-%20Panoramic%20stereo%20imaging.pdf>

<https://s3-us-west-1.amazonaws.com/disneyresearch/wp-content/uploads/20141223144328/Megastereo.pdf>

<http://richardt.name/publications/megastereo/>

<https://www.disneyresearch.com/publication/megastereo-constructing-high-resolution-stereo-panoramas/>

<https://moodle.afeka.ac.il/pluginfile.php/168615/mod_folder/content/0/cvpr99-stereopan%20-%20stereo%20panorama%20with%20a%20single%20camera%20-%20peleg%20%2B%20ben%20ezra%20-%201999.pdf?forcedownload=1>

**Appendix**

#include **<stdio.h>**#include **<iostream>**#include **<vector>**#include **<opencv/cv.h>**#include **<opencv/highgui.h>**#include **"opencv2/core.hpp"**#include **"opencv2/imgproc.hpp"**#include **"opencv2/highgui.hpp"**#include **"opencv2/core/core.hpp"**#include **"opencv2/features2d/features2d.hpp"**#include **"opencv2/highgui/highgui.hpp"**#include **"opencv2/xfeatures2d/nonfree.hpp"**#include **"opencv2/calib3d/calib3d.hpp"**#include **"opencv2/imgproc/imgproc.hpp"**#include **"opencv2/stitching.hpp"  
  
using namespace** std;  
**using namespace** cv;  
  
  
**void** interactiveMenu(**const** string videoNames[], **int** numOfVideos, **int**& chosenOption, **int**& frameFreq);  
  
**void** loadFramesAndShowVideo(**int** chosenOption, vector<Mat>& frames, **int** frameRate);  
  
**void** fixFramesSize(vector<Mat>& frames);  
  
**void** homographiesBetweenFollowingFrames(vector<Mat> &frames, vector<**int**> &dxs, vector<Mat> &homographies);  
  
**void** checkImages(**const** Mat& gray\_image1,**const** Mat& gray\_image2);  
  
vector< DMatch > detectExtractAndMatchFeatures( Ptr<Feature2D>& detector, Ptr<Feature2D>& extractor, Mat& image1, Mat& image2,  
 vector< KeyPoint >& keypoints\_object, vector< KeyPoint >& keypoints\_scene, Mat& descriptors\_object, Mat& descriptors\_scene);  
  
vector< DMatch > findGoodMatches(Mat& descriptors\_object, vector< DMatch >& matches);  
  
**void** mergeFrames(**int** positionOffset, vector<Mat> &frames, vector<**int**>& dxs, **int** sliceWidth, Mat& result, **int** height);  
  
**void** fitSize(Mat& leftEye, Mat& rightEye);  
  
**void** matToRed(Mat& inputFrame);  
  
**void** matToCyan(Mat& inputFrame);  
  
**void** createPanoramasAndAnaglyph(vector<Mat> &frames, vector<**int**> &dxs, **int** margin, **int** frameRate);  
  
**void** createShowAndWritePanorama(vector<Mat> &frames, vector<**int**>& dxs, Mat& result, **int**& range, **int** margin, **int** frameRate);  
  
  
**constexpr int** NUM\_OF\_VIDEOS = 5;  
  
string VIDEO\_PATHS[] = {**"/Users/roie/Desktop/mountain.mov"**,  
 **"/Users/roie/Desktop/gardenWithMoshe.mp4"**,  
 **"/Users/roie/Desktop/playground3.mp4"**,  
 **"/Users/roie/Desktop/insideHouse23.mp4"**,**" "**},  
  
 VIDEO\_NAMES[] = {**"Mountain"**, **"House Garden"**,**"Playground"**, **"Living Room"**,**"Your Own Video"**};  
  
string vidName;  
  
*/\*\*  
 \* Created by Roie Danino and Moshe Amini  
 \*  
 \*/***int** main()  
{  
 **int** margin = 6, frameFreq;  
 **int** chosenOption;  
  
 vector<Mat> frames;  
 vector<**int**> dxs;  
 vector<Mat> homographies;  
  
 interactiveMenu(VIDEO\_NAMES, NUM\_OF\_VIDEOS, chosenOption, frameFreq);  
  
 loadFramesAndShowVideo(chosenOption, frames,frameFreq);  
  
 fixFramesSize(frames);  
  
 homographiesBetweenFollowingFrames(frames, dxs, homographies);  
  
 createPanoramasAndAnaglyph(frames, dxs, margin, frameFreq);  
  
 waitKey(0);  
}  
  
**void** interactiveMenu(**const** string videoNames[], **int** numOfVideos, **int**& chosenOption, **int**& frameFreq)  
{  
 **bool** stopLoop = **false**;  
 string menu = **"Please choose one of the following videos: "**;  
 **int** choice = 0;  
 **for** (**int** i = 0; i < numOfVideos ; i++)  
 {  
 menu += **" \n "**+ to\_string(i+1) + **") "** + videoNames[i];  
 }  
  
 **do** {  
 **try** {  
 cout << menu << endl;  
 cin >> choice;  
  
 **if** (choice < 1 || choice > numOfVideos)  
 **throw** Exception();  
  
 **if**(choice == numOfVideos)  
 {  
 string path;  
 cout <<**"Please enter full path of your video:"**;  
 cin >> path;  
 VIDEO\_PATHS[numOfVideos - 1] = path;  
  
 }  
  
 cout <<**"Please Enter Frame Frequency in range: 1 (High Quality) - 10 (Fast)"** << endl;  
 cin >> frameFreq;  
  
 **if**(frameFreq < 1 || frameFreq > 10)  
 **throw** Exception();  
 **else** stopLoop = **true**;  
 }  
 **catch** (Exception& exception) {  
 cout << **"Oops! Try Again!"** << endl;  
 }  
 }**while**(!stopLoop);  
  
 chosenOption = --choice;  
}  
  
**void** loadFramesAndShowVideo(**int** chosenOption, vector<Mat>& frames, **int** frameRate)  
{  
 **const** string windowName = **"Input Video"**;  
 Mat image;  
 VideoCapture cap(VIDEO\_PATHS[chosenOption]);  
  
 vidName = VIDEO\_NAMES[chosenOption];  
  
 namedWindow(windowName, ***CV\_WINDOW\_FREERATIO***);  
 **int** counter = 0;  
  
 **int** fps = (**int**)cap.get(***CV\_CAP\_PROP\_FPS***);  
  
 cout << fps << endl;  
  
 **while** (**true**)  
 {  
 **if** (!cap.read(image))  
 **break**;  
  
 **if** (counter % frameRate == 0)  
 frames.push\_back(image.clone());  
  
 imshow(windowName, image);  
  
 waitKey(fps);  
  
 counter++;  
 }  
 destroyWindow(windowName);  
 image.release();  
 cap.release();  
}  
  
**void** fixFramesSize(vector<Mat>& frames)  
{  
  
 **for** (**int** i = 0; i < frames.size(); i++)  
 {  
 **if** (frames.at(i).cols > 1000 && frames.at(i).rows > 1000)  
 resize(frames.at(i), frames.at(i), Size(frames.at(i).rows / 8, frames.at(i).cols / 8));  
 }  
}  
  
**void** homographiesBetweenFollowingFrames(vector<Mat> &frames, vector<**int**> &dxs, vector<Mat> &homographies)  
{  
 Ptr<Feature2D> detector = xfeatures2d::SIFT::create();  
 Ptr<Feature2D> extractor = xfeatures2d::SIFT::create();  
 vector< DMatch > matches;  
 Mat image1,image2,gray\_image1, gray\_image2, descriptors\_object, descriptors\_scene,H;  
  
 **for** (**int** j = 0; j < frames.size() - 1; j++)  
 {  
 cout << j << endl;  
 image1 = frames.at(j + 1);  
 image2 = frames.at(j);  
  
 cvtColor(image1, gray\_image1, ***CV\_RGB2GRAY***);  
 cvtColor(image2, gray\_image2, ***CV\_RGB2GRAY***);  
  
 checkImages(gray\_image1, gray\_image2);  
  
 vector< KeyPoint > keypoints\_object, keypoints\_scene;  
  
 matches = detectExtractAndMatchFeatures(detector,extractor,gray\_image1,gray\_image2,keypoints\_object,keypoints\_scene,descriptors\_object,descriptors\_scene);  
  
 vector<DMatch> good\_matches = findGoodMatches(descriptors\_object, matches);  
  
 std::vector< Point2f > obj;  
 std::vector< Point2f > scene;  
  
 cout << **"good matches: "** + to\_string(good\_matches.size()) << endl;  
  
 **for** (**int** i = 0; i < good\_matches.size(); i++)  
 {  
 obj.push\_back(keypoints\_object[good\_matches[i].queryIdx].pt);  
 scene.push\_back(keypoints\_scene[good\_matches[i].trainIdx].pt);  
 }  
  
 H = findHomography(obj, scene, **CV\_RANSAC**);  
  
 cout << H << endl;  
  
 **int** dx = (**int**)(H.at<**double**>(Point(2, 0)));  
 **if** (dx < 0)  
 dx \*= -1;  
  
 dxs.push\_back(dx);  
 homographies.push\_back(H.clone());  
 }  
  
 image1.release();  
 image2.release();  
 gray\_image1.release();  
 gray\_image2.release();  
 descriptors\_object.release();  
 descriptors\_scene.release();  
 H.release();  
  
}  
  
**void** checkImages(**const** Mat& gray\_image1,**const** Mat& gray\_image2)  
{  
 **if** (!gray\_image1.data || !gray\_image2.data)  
 {  
 std::cout << **" --(!) Error reading images "** << std::endl;  
 exit(1);  
 }  
}  
  
vector< DMatch > detectExtractAndMatchFeatures( Ptr<Feature2D>& detector, Ptr<Feature2D>& extractor, Mat& image1, Mat& image2,  
 vector< KeyPoint >& keypoints\_object, vector< KeyPoint >& keypoints\_scene, Mat& descriptors\_object, Mat& descriptors\_scene)  
{  
 detector->detect(image1, keypoints\_object);  
 detector->detect(image2, keypoints\_scene);  
  
 extractor->compute(image1, keypoints\_object, descriptors\_object);  
 extractor->compute(image2, keypoints\_scene, descriptors\_scene);  
  
 BFMatcher matcher;  
 vector< DMatch > matches;  
  
 matcher.match(descriptors\_object, descriptors\_scene, matches);  
  
 **return** matches;  
}  
  
vector< DMatch > findGoodMatches(Mat& descriptors\_object, vector< DMatch >& matches)  
{  
 **double** max\_dist = 0;  
 **double** min\_dist = 100;  
  
 **for** (**int** i = 0; i < descriptors\_object.rows; i++)  
 {  
 **double** dist = matches[i].distance;  
 **if** (dist < min\_dist) min\_dist = dist;  
 **if** (dist > max\_dist) max\_dist = dist;  
 }  
  
 printf(**"-- Max dist : %f \n"**, max\_dist);  
 printf(**"-- Min dist : %f \n"**, min\_dist);  
  
 vector<DMatch> good\_matches;  
 **int** good = 0;  
 **double** count = 2;  
 cout <<**"Matches: "** << descriptors\_object.rows << endl;  
 **while** (good < 50)  
 {  
 good\_matches.clear();  
 **for** (**int** i = 0; i < descriptors\_object.rows; i++)  
 {  
 **if** (matches[i].distance <= count \* (min\_dist + 1))  
 {  
 good\_matches.push\_back(matches[i]);  
 }  
 }  
 good = good\_matches.size();  
 count += 0.5;  
 }  
 **return** good\_matches;  
}  
  
**void** createShowAndWritePanorama(vector<Mat> &frames, vector<**int**>& dxs, Mat& result, **int**& range, **int** margin, **int** frameRate)  
{  
 frames.at(0).copyTo(result(Rect(0, 0, frames.at(0).cols, frames.at(0).rows)));  
 **for** (**int** i = 0; i < dxs.size(); i++)  
 {  
 Mat temp = frames.at(i + 1).clone();  
 Mat temp2 = temp(Rect(temp.cols - dxs.at(i), 0, dxs.at(i), temp.rows));  
 temp2.copyTo(result(Rect(frames.at(0).cols + range, 0, temp2.cols, temp.rows)));  
 range += dxs.at(i) - margin;  
 temp.release();  
 }  
 imshow(**"Panorama"**, result);  
 imwrite(**"/Users/roie/Desktop/panorama "**+ vidName + to\_string(frameRate) + **" .jpg"**, result);  
}  
  
**void** mergeFrames(**int** positionOffset, vector<Mat> &frames, vector<**int**>& dxs, **int** sliceWidth, Mat& result, **int** height)  
{  
 **int** atFrame, howMuch, range = sliceWidth;  
 **for** (**int** i = 1; i < frames.size(); i++)  
 {  
 **if** (dxs.at(i - 1) < sliceWidth)  
 {  
 atFrame = positionOffset + sliceWidth - dxs.at(i - 1);  
 howMuch = dxs.at(i - 1);  
 }  
 **else** {  
 atFrame = positionOffset;  
 howMuch = sliceWidth;  
 }  
 Mat temp = frames.at(i)(Rect(atFrame, 0, howMuch, height));  
 temp.copyTo(result(Rect(range, 0, howMuch, height)));  
 range += howMuch;  
 }  
}  
  
**void** fitSize(Mat& leftEye, Mat& rightEye)  
{  
 **if** (leftEye.cols <= rightEye.cols)  
 {  
 rightEye = rightEye(Rect(0, 0, leftEye.cols, rightEye.rows));  
 }  
 **else** {  
 leftEye = leftEye(Rect(0, 0, rightEye.cols, leftEye.rows));  
 }  
}  
  
**void** matToRed(Mat& inputFrame)  
{  
 *//Only Red* **for** (**int** i = 0; i < inputFrame.cols; i++)  
 {  
 **for** (**int** j = 0; j < inputFrame.rows; j++)  
 {  
 (inputFrame.at<Vec3b>(Point(i, j))).val[0] = 0;  
 (inputFrame.at<Vec3b>(Point(i, j))).val[1] = 0;  
 }  
 }  
}  
  
**void** matToCyan(Mat& inputFrame)  
{  
 *//Only Blue and Green - cyan* **for** (**int** i = 0; i < inputFrame.cols; i++)  
 {  
 **for** (**int** j = 0; j < inputFrame.rows; j++)  
 {  
 (inputFrame.at<Vec3b>(Point(i, j))).val[2] = 0;  
 }  
 }  
}  
  
**void** createPanoramasAndAnaglyph(vector<Mat> &frames, vector<**int**> &dxs, **int** margin, **int** frameRate)  
{  
 **int** total\_dxs = 0;  
 **for** (**int** i = 0; i < dxs.size(); i++)  
 {  
 total\_dxs += dxs.at(i);  
 }  
  
 Mat result(frames.at(0).rows, frames.at(0).cols + total\_dxs, 16);  
 **int** range = -margin;  
  
 createShowAndWritePanorama(frames, dxs, result, range, margin, frameRate);  
  
 Mat firstFrame = frames.at(0);  
 Mat panorama = result;  
   
 **int** width = firstFrame.cols, height = firstFrame.rows;  
  
 **int** sliceWidth = width / 7;  
   
 **int** leftPosition = 2 \* sliceWidth;  
 **int** rightPosition = sliceWidth;  
  
 **int** widthLeft = total\_dxs + sliceWidth, widthRight = total\_dxs + sliceWidth;  
   
 Mat rightEye(height, widthRight, 16), leftEye(height, widthLeft, 16);  
  
 mergeFrames(rightPosition, frames, dxs, sliceWidth, rightEye, height);  
  
 imshow(**"Right"**, rightEye);  
 imwrite(**"/Users/roie/Desktop/right "**+ vidName+ to\_string(frameRate) +**" .jpg"**, rightEye);  
   
 mergeFrames(leftPosition, frames, dxs, sliceWidth, leftEye, height);  
 imshow(**"Left"**, leftEye);  
 imwrite(**"/Users/roie/Desktop/left "**+ vidName + to\_string(frameRate) + **" .jpg"**, leftEye);  
   
 rightEye = rightEye(Rect(leftPosition - rightPosition, 0, rightEye.cols - (leftPosition - rightPosition), rightEye.rows));  
  
 fitSize(leftEye, rightEye);  
  
 matToCyan(rightEye);  
  
 matToRed(leftEye);  
  
 Mat stereo = leftEye + rightEye;  
   
 imwrite(**"/Users/roie/Desktop/stereo "** + vidName + to\_string(frameRate) + **" .jpg"**, stereo);  
   
 imshow(**"Stereo"**, stereo);  
}