Stereo Nvidia Jetson Nano Competition

Bruno Principe
DEI
University of Padua, Veneto
Padua, Italy
bruno.principe@studenti.unipd.it

Luca Sambin
DEI
University of Padua, Veneto
Padua, Italy
luca.sambin@studenti.unipd.it

***Abstract*—In this work we are focusing on designing and deploy the Stereo Matching (SM) approach that has an acceptable trade-off between the fastest and the most accurate one. To come with this aim we implemented 2 algorithm, the classical Census Semi-Global Matching and a Hybrid algorithm that exploit the acceleration and low memory consumption of the traditional approaches and the learning methods as Deep Learning (DL) what is termed StereoVAE.**

***Index Terms*—Stereo Matching, Competition, Embedding Systems, Nvidia Jetson Nano**

I. Introduction

Stereo Matching is an important requirement for many applications, as instance 3D reconstruction, robot-navigation, driveless cars and so on. Despite this its usability is still limited because attaining high accuracy requires a very high computational complexity. Stereo matching methods take a pair of left-right images from stereo cameras as input and generate the disparity map for depth estimation. Typical stereo matching algorithms can be classified into two categories: local and global methods. Additional practical problems originate from recording and illumination differences or reflections, because matching is often directly based on intensities that can have quite different values for corresponding pixels. Now in summary we can say that with the assumption of a simple translation between the cameras (otherwise, images must be rectified using multiple extrinsic and intrinsic camera parameters), the corresponding points must be in the same row of both images, along the epipolar lines. A similarity measure correlates matching pixels and the disparity (d) is the similarity distance between both points. The high computational load and memory bandwidth requirements of SGM pose hard challenges for fast and low energy-consumption implementations. Dedicated hardware solutions (e.g. FPGA or ASIC) achieve these goals, but they are very inflexible regarding changes in the algorithms. Implementations on desktop GPUs can assure real-time constraints , but their high power consumption and the need to attach a desktop computer makes them less suitable for embedded systems. Deep networks can solve the matching problem via supervised learning in an end-to-end fashion, and they have the ability to incorporate local context as well as prior knowledge into the estimation process. On the other hand, deep neural networks tend to be computationally intensive and suffer from significant latency when processing high-resolution stereo images. Thus far, many researchers have focused upon accelerating stereo matching on mobile platforms. Most of them focus on accelerating the two most computationally intensive stages: cost calculation and cost aggregation. During cost calculation, each pixel in the reference image is first matched with several pixels in the target image one by one. Next, the similarity between any two pixels is quantified by a numerical value (cost) calculated by a matching method, such as the sum of absolute differences (SAD), census, or convolution neural network (CNN). Various combinations of the above two stages not only result in different matching accuracies, but also different computational complexities. We present simple, but well-designed, Fast GPU SGM baseline and we propose a fast trade-off stereo-matching frame that combines the advantages of traditional and CNN-based methods. The remainder of this paper is organized as follows: Section II presents the generation methods for a low-resolution disparity image and StereoVAE structure. The experimental results are discussed in section III. Section IV summarizes our work and presents a future research direction.

II. Methods

In this section we introduce the 2 blocks that makes the StereoVAE routine: A Traditional SM algorithm and a custom Varietional Autoecoder. Inspired by the work of [1] the pipeline is represented below:

- Census Transform Hamming Distance Cost: indicates the relative order of intensities in a local window centered at the pixel of interest, not the intensity values themselves. It encodes each window into a bit string as follows:

  $$CT(p) = \otimes q \in Np\Phi(I(p), I(q))$$

  where $\otimes$ denotes the bit-wise concatenation, and I(p) and I(q) are the values of the central pixel and the neighboring pixel in the window Np , respectively. $\Phi(i, j)$ is set to 1, if $i > j$, or 0 otherwise. After encoding, the matching cost is calculated through the Hamming distance between bit strings of corresponding pixels in the image pair, which is defined as the number of bits that are not equal, as shown below:

  $$C_{census}(p, d) = H(CTb(p), CTm(pd))$$

  where CTb (p) and CTm (p-d) are transformed bit strings of corresponding pixels in the image pair.

- A general Census SGM method solves a one-dimensional minimization problem, along different paths r=(rx, ry) using the recurrence and a dynamic programming algorithmic pattern. Matrix L_r contains the smoothing aggregated costs for path r. The first term is the original matching cost and the second term adds the minimum cost of the disparities corresponding to the previous pixel (x−rx, y−ry), including penalties for small disparity changes (P1) and for larger disparity discontinuities and (P2). P1 is intended to detect slanted and curved surfaces, while P2 smooths the results and makes abrupt changes difficult. The last term ensures that aggregated costs are bounded. The different Lr matrices must be added together to generate a final cost.
- Disparity Correction: Based on the aggregation results, the disparity map can be generated using $D_{map}(x, y) = argmin(C_{Final}(x, y, d))$, d where C_{Final} (x, y, d) represents the final matching costs aggregated. Then by exploiting a median filter with a user-defined window size to remove outliers and smooth the disparity map.
- StereoVAE: It consists of feature extraction, VAE, and upscaling modules; VAE can be subdivided into encoder and decoder units. Our StereoVAE receives two types of input images: one is the original left image, and the other is the disparity map generated using traditional methods in the previous step. Both are a downsampled of 200x300 size with the respect of the original image's size. The output is a high-resolution disparity map amplified by our network. X_0 and X_1 represent the low-resolution disparity map and the grayscale left image as feature-extraction (FE) module inputs, respectively. These two inputs are combined to extract the boundary correspondence information between the disparity map and the left image because disparity changes are usually drastic in these regions. The objective of the VAE module is to reconstruct the features extracted by the FE module; the inputs FE_1 and FE_2 of the encoder are the outputs of the FE module. The outputs of the encoder module second layer are downsampled to two dimensions, mu and sigma, using a convolutional kernel of 1×1. mu denotes the mean of a normal distribution, and sigma denotes the variance logarithm of a normal distribution. The training target of the VAE is to obtain a standard normal distribution whose mean is 0 and variance is 1. The input of the decoder module is a normal distribution with the mean and variance increased of 32 dimensions by means of combinations of deconvolutional layers and residuals blocks. the combination of the reduced disparity input the concatenated features and the decoder outcome is then upsampled twice to lead to the final disparity map. The model use the Leaky Relu activation function with slope $0.1x$ for $x < 0$ and x for $x > 0$. The StereoVae loss function is simply the sum of the Mean Absolute Loss (MAL) and a custom $KDLoss = 0.5 * \{(mu - 1)^2 1 + e^{(sigma-1)^2} \log e^{(sigma-1)^2}\}$.

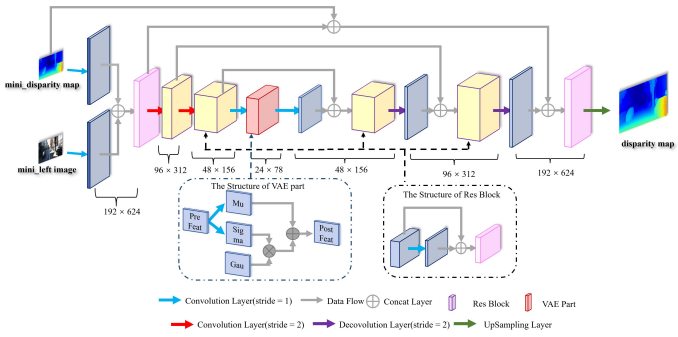

Fig. 1. StereoVAE Structure [1]

FEATURE EXTRACTION						
Layer	Input	Input Size	Output	Output size	Output Kernel Number	Kernel Size
1	X_0	H/2 * W/2	FE_1	H/2 * W/2	32	5 * 5
1	X_1	H/2 * W/2	FE_2	H/2 * W/2	32	5 * 5
ENCODER						
Layer	Input	Input Size	Output	Output size	output kernel number	Kernel Size
1	FE_1 & FE_2	H/2 * W/2	EN_1	H/4 * W/4	16	5 * 5
1	EN_1	H/4 * W/4	EN_1_R	H/4 * W/4	8	5 * 5
2	EN_1 & EN_1_R	H/4 * W/4	EN_2	H/8 * W/8	32	5 * 5
2	EN_2	H/8 * W/8	EN_2_R	H/8 * W/8	16	5 * 5
2	EN_2 & EN_2_R	H/8 * W/8	mu & sigma	H/8 * W/8	2	1 * 1
DECODER						
Layer	Input	Input Size	Output	Output size	output kernel number	Kernel Size
1	sigma * gaussian + mu	H/8 * W/8	D_1	H/8 * W/8	32	1 * 1
1	D_1 & EN_2 & EN_2_R	H/8 * W/8	D_1_R	H/8 * W/8	16	5 * 5
2	D_1 & EN_2 & EN_2_R & D_1_R	H/8 * W/8	D_2	H/4 * W/4	16	5 * 5
2	D_2 & EN_1 & EN_1_R	H/4 * W/4	D_2_R	H/4 * W/4	8	5 * 5
3	D_2 & EN_1 & EN_1_R & D_2_R	H/4 * W/4	D	H/2 * W/2	16	5 * 5
UP-SAMPLING						
Layer	Input	Input Size	Output	Output size	output kernel number	Kernel Size
1	D & X_0 & FE_1 & FE_2	H/2 * W/2	Y	H * W	1	5 * 5

Fig. 2. Summary of the parameter of the StereoVAE Structure [1]

A. Semi Global Matching Cuda implementation

Since the required operations are very time-consuming, classical CPU implementations are not real-time capable. Thus due the hardware limitation of the Nvidia Jetson Nano 2gb Developer Kit both in terms of memory and velocity of the CPU operation we focused in the CUDA library that allows programming graphics hardware in C with extensions for parallel functions, data transfer, and explicit caching. CUDA follows a SIMD (Single Instruction Multiple Data) execution model enabling the programmer to frame a computational task (kernel) that is performed in the same manner on small chunks of data. These tasks are assigned to lightweight threads executed on the graphics hardware in parallel. The threads of a kernel function are organized in a two-level hierarchical structure with up to three dimensions. The grid is divided into several blocks holding the threads. Generally, each thread is assigned data (e. g., a single pixel of an image) depending on its location within the grid / block. The following implementation can be encapsulated into 3 huge operations : Initial cost computation , that here we can rephrased as compute the Census Transform, the cost aggregation and the post processing on the disparity.

1) Center Symmetric Census Transform: A 9×7-window, Center-Symmetric Census Transform (CSCT) concatenates the comparisons of 31 pairs of pixels into a bit-vector feature. Only center-symmetric pairs of pixels are compared, but over an image patch of n × m. The gained bits may be used to encode a weighted Hamming Distance through bit duplication. This fits well to implementations using hardware bit count

Info	CensusSGM Reduced	VPISGM	SGM2016
Time	**51.8676 ms**	75.980456 ms	88.3338467 ms
bad_2_0	67.4294147	37.2437333	**36.5268521**
FPS	**18.7726027 fps**	17.79852112 fps	11.6147873 fps

TABLE I

AVERAGE EXECUTION TIME, AVERAGE BAD 2.0 AND AVERAGE FPS IN THE MIDDELBURY EVALUATION DATASET : OUR CENSUSSGM WITH REDUCED SIZE IMAGES AND 64 DISPARITY SIZE WITH PARAMETER [P1=10, P2=120, SUBPIX=FALSE, 4 PATHS, SYMMETRIC CENSUS] , VPI LIBRARY VERSION OF SGM AND [3] SGM2016 WITH PARAMETER [SIZE DISPARITY = 128, SIZE OF THE IMAGE = KITTI IMAGE SIZE, P1=7 , P2=84]

instructions for the Hamming Distance. It is defined with the following equation :

$$CS - CT_{m,n}(x,y) = \bigotimes_{(i,j)\in L} s(I(xi,yj), I(x+i,y+j))$$

with $L = L_1 \bigcup L_2$, $L_1 = R_{n'_0,0} \times R_{m'_0,0} \setminus\{(0,0)\}$, $L_2 = R_{1,n'} \times R_{m',1}$ and $R_{a,b} = \{x \in Z | a \leq x \leq b\}$. Where the image patch is of n × m pixels. In the implementation was opted to preserve two modality of execution, this CSCT version and the traditional CT, now depending also on the input image format, that could be or 8UInt or 16UInt and the output image incapsulated into a 32UInt format. Apart from these not so relevant variations, the parallelization of the CSCT works as follow : after defined the half size of the height and width of the block window and the shared memory buffer size, we are going to save the value of the source image to a temporary shared memory matrix at position i of the current window and the current thread index of x only if (0 ¡= x and x ¡ width) and (0 ¡= y and y ¡ height), then synchronize and read the data from the shared memory. From here is unfolded the for loop and processed synchronously the computation of the Census Transform till the current index is between (- half of the width size of the window) and (block size - half width size of the window). Now for the CT the pattern is similar with a single difference instead of assign the same value computed to the current id to the symmetric id, we compute it for all the windows position. For the pseudocode look at Algorithm 1.

2) Cost Aggregation: Now that we collected the Census transformation with the Hamming distance the next step we aggregate the cost differentiating on the maximum size of the disparity : 64, 128 or 256 and in base of the activation of the SubPixel modality for which multiply to 16 bit the disparity end up to having a 64 bit Census transform result with the respect to the 32 Bit Census Transform in case SubPixel mode is set to false. The execution of this task depends also in the number of paths chosen from 4 and 8, in the first case it includes just the vertical and horizontal motion in both the direction (from up to down and from down to up, same for right to left and left to right), but if the second case is selected then it performs also the diagonal motion in both

Algorithm 1: CSCT parallel CUDA

Data: src, w, h, pitch
Result: dst

for *from* $i = 0$ *to* $i \prec h$ **do**
  $x = x_0 + threadId, y = y_0 - (h/2) + i;$
  if $0 \leq x \prec w$ *and* $0 \leq y \prec h$ **then**
    $val = src[x + y * pitch];$
  else
    $val = 0;$
  $SharedM[i][threadId] = val;$
CSCT Barrier Synchronization;
Unrolled parallel **for** *from* $i = 0$ *to* $(lines\_per\_block = 16)$ **do**
  if $i + 1 \prec lines\_per\_block$ **then**
    $SharedM[smem_y][smem_x] = val;$
  if $(w/2) <= threadId \prec block\_size - (w/2)$ **then**
    if $(w/2) \leq x \prec w - (w/2)$ *and* $(h/2) \leq y \prec h - (h/2)$ **then**
      $feat = 0;$
      for *from* $dy = -(h/2)$ *to* $dy \leq (h/2)$ **do**
        for *from* $dx = -(w/2)$ *to* $dx \leq (w/2)$ **do**
          feat = ($feat \prec\prec 1$*)* | (*SharedM[*$smem_y$*]*$[smem_x] \succ$ $SharedM[smem_y 1][smem_x 1]$*);*
      $dst[x + y * w] = f;$
  CSCT Barrier Synchronization;

the directions. The clue of the speed in the computation of the disparity stands in creating a stream for each path to be executed and finally synchronize the streams or in other words run them in parallel. We list just 3 aggregation Kernels for describing the horizontal, vertical and oblique motion handling both the direction. To generalize an aggregation Kernel first define the Disparity Census right image values buffer, then taking into account the block division put its values to the shared memory accessible to all the threads, with the Read-Only data cache load function the disparity Census left image data is read, then compute the local cost = COUNT_ONE_BITS(l_value XOR l_right). Subsequently update the global cost that rely in 3 steps : initialize the Lazy output = min(min(disparity[0] - last_min, p2), disparity[0 + 1] - last_min + p1) + local_costs[0], then update either the local minimum and the lazy output for all the block size and finally compute the final minimum cost that termed as last_min. Only at the end as last operation is stored the final disparity estimated.

3) Disparity's Post Processing: The disparity found is raw, Hence it undergoes to 4 Post-Processing steps :

1) Winner-Take-All (WTA)
2) Median Filter (MF)

Algorithm 2: Cost Aggreagation Kernel Schema CUDA

Data: left, CENSUS_TYPE right, w, h, p1, p2, min_disp
Result: dest

initialize warp_id, group_id , lane_id and shfl_mask;
/* Load Right Disparity from the Shared Memory */
parallel **for** *from* $i = 0$ *to* $i \prec h$ **do**
  for *from* $i_0 = 0$ *to* $i_0 \prec RIGHT\_BUFFER\_SIZE$ *with* $i_0 += BLOCK\_SIZE$ **do**
    $i = i_0 + threadIdx.x$;
    if $i \prec RIGHT\_BUFFER\_SIZE$ **then**
      $right\_x = right\_x_0 + PATHS\_PER\_BLOCK - 1 - i - min\_disp$;
      CENSUS_TYPE right_value = Load($right[y * w], right\_x, w$);
      $lo = i\%DP\_BLOCK\_SIZE$;
      $hi = i/DP\_BLOCK\_SIZE$;
      $right\_buffer[lo][hi] = right\_value$;
      if ($hi \succ 0$) $right\_buffer[lo + DP\_BLOCK\_SIZE][hi - 1] = right\_value$;
  CSCT Barrier Synchronization;
  /* Aggregate the cost */
  if $0 \leq x \prec w$ **then**
    left_value = Load(left[x + y * w]);
    right_values[DP_BLOCK_SIZE];
    for *from* $j = 0$ *to* $j \leq DP\_BLOCK\_SIZE$ **do**
      right_values[j] = right_buffer[right0_addr_lo + j][right0_addr_hi];
    local_costs[DP_BLOCK_SIZE];
    for *from* $j = 0$ *to* $j \prec DP\_BLOCK\_SIZE$ **do**
      local_costs[j] = COUNT_BITS(left_value XOR right_values[j]);
    Update_Disp(local_costs, p1, p2, shuffle_mask);
  CSCT Barrier Synchronization;

3) Consistency Check (CC)
4) Correction of Disparity Range (CDR)

In WTA the CUDA kernel tails this routine : compute the sum of the cost parallelizing and unrolling the loop, and guarantees memory ordering among threads participating in the barrier. Next load the sum of costs , in order to reduce the execution time the sum of cost is encapsulated in a dictionary along with the disparity associated. After it the left and the right disparity are all updated in order to store the best disparity value and in the end if then we check if the value is or not unique in two step :

1) check if $Cost * uniquenness \geq BestCost$
2) check if $current\_idx - best\_idx \leq 1$

If one of the above constraints are satisfied shuffle the data under the bitwise XOR, if is true then the disparity is valid otherwise the value is substituted by an invalid disparity value. For the right disparity instead just store the best disparity value. The obtained temporary disparities are then smoothed under the MF, with a kernel size k = 3 and a radius = k/2, so we have a window of size 9 and in the range of the radius we collect in the buffer all the disparity value that lies in the window and compute the median and stores it to the disparity in position y * step_size + x. If the current value does not be in the in the radius range is set to 0. In CC in a parallelized way was checked the Left Disparity consistency, picked the source left image, the left and right disparity computed until now the invalidity of a disparity value is confirmed if already is in the previous step or in the source image or in case the the constant acceptance difference pixel accdiff is given and $disp\_l\_value - disp\_r\_value \succ accdiff$ then the current value us set to invalid. The final step require to perform the parallelized CDR routine that naively assign the global invalid value: $(min\_disp\_range - 1) * scale$, in case the range of the disparity is out of the maximum disparity range.

B. StereoVae Inputs Pre-Processing

In order to be processed to a Deep Learning model the two input that are the left reduced gray scale image and the resulting left Disparity Map must undergoes just a Normalization step that requires only to be converted into a 32 Float Bit normalizing on the full disparity range to have values from 0 to 1, then transform them into Tensors. The model functionality was described above and for a figurative representation look at Image 1 and Image 2.

1) Training Dataset: For creating the custom built Training, Validation and Test data just run the the Census SGM in reduced mode as described above and for other details in the next paragraph and save the image in classic Image format as PNG or JPG. Collects also the Left Ground Truth (GT) of the dataset to be used as a supervised label during the training, and resize it as $396x596$, normalize it to constraints to have values 0 to 1 with float format and trasform it to Tensor.

III. Experiments

The testing are all performed in the Nvidia Jetson Nano architecture, and all the library already precluded in the original set-up of the machine was used : CUDA, and TensorRT. Only the Python implementation of the model that is easily accessible from the Jupyter Notebook is implemented in Pytorch and the library used can be installed from it, in fact for training the model we opted to do it in a Desktop environment that allowed a better sized batch , in our case equal to 16. The difference with the original work we switched to use the classical Census Trasformation SGM to boost in

the performance with the CUDA implementation, and we quantized (Traslated from Pytorch to ONNX then corrected it with PolyGraphy Sanitized and finally TrTExec to obtain the engine and test it) the final model to be used as inference with the TensorRT library to be used with the StereoVae C++ CUDA script. Another variations with the respect to the original paper is the training with a synthetic dataset such as Sintel for training and validation and test directly with all the other public available SM datasets such as Kitti2012 and InK2022. The full test was done with the CUDA Census Hamming SGM models (CensusSGM) with the respect to the other SGM baselines from VPI library version of SGM and [3] SGM that we referred as SGM2016 in this paper. The first test done was an ablation study of the CensusSGM by modifying the number of directions to compute the disparity map the option can be only 4 or 8, the type of Census approach from which we can list the traditional windowed Census transform with a window of 9x7 and a symmetric version of the traditional Census. In addition to that the Subpixels accuracy modality was tested with the respect to the absence to it, the maximum disparity for which can be only 64 or 128 and the size of the processed image. In this last case the image can be in the original size (Full modality) or reduced to a size of 300x200 (Reduced modality), the reduced size was decided as a trade off between processing velocity and image's details quality in both the disparity computation of CensusSGM and its upscaling and correction with StereoVae model. In case was adopted the Reduced modality of CensusSGM, without the execution of StereoVae model, after obtained the disparity map then as post-processing its values are re-scaled basing on how much the image was downscaled with the respect to the original size. As instance if the image was reduced to $1/2$ then the final disparity values are multiplied as disparity = disparity * 2 , then changed format to Float32 bit and finally stored as PFM format to be evaluated with the Middlebury Benchmark. Due to the constraints of SGM2016 that was engineerized to run only images of the Kitti size and the output disparity was only in the U8Int format, so the image is downscaled to fit the Kitti's image size and as post-processing the disparity map is converted to 32Float Bit, Upscaled with the Nearest Neighboor Interpolation in order to avoid the contamination of the invalid values in the upscaled disparity map and at the end the disparity values are multiplied to the width size difference value as done in CensusSGM to correct them. The VPISGM approach instead just need the conversion from the 16SInt to 32Float format, same for CensusSGM in case was selected the "Full" size processing modality. To note that during the ablation study we carry on using the penalization term P1 = 10 , and the smoothing term P2 = 120 focusing on how much the accuracy and the execution time changes for each of the parameter chosen. Beside this the CensusSGM's ablation test cases are listed below :

- Full size image with 128 Disparity size, 4 Paths, No Subpixels accuracy and P1=10, P2=120 and symmetric Census.
- Full size image with 128 Disparity size, 8 Paths, No Subpixels accuracy and P1=10, P2=120 and symmetric Census.
- Full size image with 128 Disparity size, 4 Paths, with Subpixels accuracy and P1=10, P2=120 and symmetric Census.
- Full size image with 128 Disparity size, 8 Paths, with Subpixels accuracy and P1=10, P2=120 and symmetric Census.
- Reduced size image with 128 Disparity size, 4 Paths, No Subpixels accuracy and P1=10, P2=120 and symmetric Census.
- Reduced size image with 128 Disparity size, 8 Paths, No Subpixels accuracy and P1=10, P2=120 and symmetric Census.
- Reduced size image with 128 Disparity size, 4 Paths, with Subpixels accuracy and P1=10, P2=120 and symmetric Census.
- Reduced size image with 128 Disparity size, 8 Paths, with Subpixels accuracy and P1=10, P2=120 and symmetric Census.
- Full size image with 64 Disparity size, 4 Paths, No Subpixels accuracy and P1=10, P2=120 and symmetric Census.
- Full size image with 64 Disparity size, 8 Paths, No Subpixels accuracy and P1=10, P2=120 and symmetric Census.
- Full size image with 64 Disparity size, 4 Paths, with Subpixels accuracy and P1=10, P2=120 and symmetric Census.
- Full size image with 64 Disparity size, 8 Paths, with Subpixels accuracy and P1=10, P2=120 and symmetric Census.
- Reduced size image with 64 Disparity size, 4 Paths, No Subpixels accuracy and P1=10, P2=120 and symmetric Census.
- Reduced size image with 64 Disparity size, 8 Paths, No Subpixels accuracy and P1=10, P2=120 and symmetric Census.
- Reduced size image with 64 Disparity size, 4 Paths, with Subpixels accuracy and P1=10, P2=120 and symmetric Census.
- Reduced size image with 64 Disparity size, 8 Paths, with Subpixels accuracy and P1=10, P2=120 and symmetric Census.
- And Reduced size image with 64 Disparity size, 4 Paths, No Subpixels accuracy and P1=10, P2=120 and traditional Census.

For the overall tables of the test done the reader can refer to the V-C where are listed all the statistics of the execution of the algorithms for the Middlebury dataset, plus the full ablation study Benchmarks.

A. Ablation Study

At the end of the tests of all the parameter settings of the implementation of CensusSGM was quite expected to have

Info	CensusSGM Full	CensusSGM Reduced
Time	328.5366 ms	**51.8676 ms**
bad_2_0	**39.95266**	67.4294147
FPS	3.71269867 fps	**18.7726027 fps**

TABLE II
AVERAGE BAD 2.0 ERROR, AVERAGE EXECUTION TIME AND AVERAGE FPS OF THE FULL RESULT IN THE MIDDELBURY EVALUATION DATASET : CENSUSSGM WITH FULL SIZE IMAGES WITH 128 DISPARITY SIZE AND CENSUSSGM WITH REDUCED SIZE IMAGES AND 64 DISPARITY SIZE WITH PARAMETER [P1=10, P2=120, SUBPIX=FALSE, 4 PATHS, SYMMETRIC CENSUS] FOR BOTH

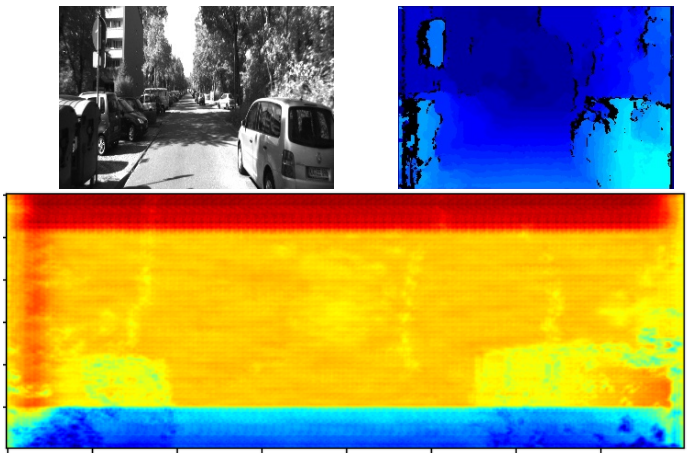

Fig. 3. Kitti2012 Disparity Map sample: the top left the original left image , the top right is the reduced left disparity map computed with CensusSGM with parameters [DISPARITY SIZE = 64, P1=10, P2=120, SUBPIX=FALSE, 4 PATHS, SYMMETRIC CENSUS] and bottom centre is the StereoVae disparity map corrected.

better disparity in case the image is processed with its own size with the respect to a low resolution counterpart, in spite of this the difference from the 128 max disparity size, 4 paths and no SubPixels accuracy run without much difference from its variations in terms of accuracy. So this means that if we use CensusSGM with the above listed parameter and the version with 8 Path and SubPixel accuracy the only difference is the time of execution for which has more weight in its application use given an example with Adirondack in the first case it takes 517.773 ms with 1.93135 fps and in the second 405.67 ms and 2.46506 fps but in both the case the Bad 2.0 error remain the same = 30.6703. If we use again the full modality but just cut the disparity size to 64 with 4 path and no SubPixel accuracy the average weighted error is reduced slightly with the respect to the 128 disparity size case $(8.62903 \prec 8.64672)$. Not only that, during the full experiments the accuracy method that prevail among the other is only the 128 standard configuration stated in the previous lines. When all comes to a execution time perspective then the case in which the image was processed with the maximum available disparity size and the original size of the image then these ones are penalized running with an average of 328.5366 ms and 3.71269867 with the respect to the average execution time of 51.8676 ms and 18.7726027 fps of the version with parameter [Disparity size = 64, P1=10, P2=120, No SubPixel accuracy, 4 paths, symmetric census] , look at Table II, and in the case of Adirondack image reach a time of 630.187 ms for the configuration [Disparity size = 128, 8 Paths , No SubPixel accuracy, P1=10, P2=120, Symmetric Census]. By replicating the testing on the 64 disparity size versions we can confirm that there is no improvement in the accuracy term with the respect to the standard parameter set-up, neither by changing the Census type from the symmetric to the classic. To draw the conclusion the fast approach has the parameter set as [Disparity size = 64, P1=10, P2=120, No SubPixel accuracy, 4 paths, symmetric census] and this is the one selected to the final assessment with the other algorithm.

B. Algorithm Statistics

If we look at the final statistics for the 3 approaches we can conclude that SGM2016 has the better accuracy in all the accuracy adopted (Bad 0.5, Bad 2.5 etc) but losing in the execution time with an average of 88.3338467 ms ,Table I. Without providing the better in accuracy or speed VPISGM represent a trade of between the accuracy of SGM2016 and the velocity of CensusSGM Reduced. In fact our CensusSGM Reduced is more suitable for an online application that the timing and fps reached during the execution in the case of the lowest resourced embedded device edition of the Nvidia Jetson Nano with 2gb it's not bad at all reaching the maximum of 25.0257 fps and a fps average of 18.7726027 fps and a minimum timing of 39.959 ms with an average of 51.8676 ms ,Table I, in the Vintage image of the Middlebury dataset.

1) Stereovae Statistics: From the execution time statistics the whole routine of StereoVae require the execution of CelsusSGM algorithm, described in details in the previous chapter, and the StereoVae model. To taking the sum of the whole performance we summed the execution time of both the processes and averaging FPS of both the algorithm. Taking into consideration the compatibility of Pytorch with TensorRT 8.2 used in the Nvidia Jetson Nano of 2gb now at End of Life (EOL), the lowest performances with the respect of the current standard of the latest Jetson Embedded devices, like Jetson Orin Nano 4gb with 20TOPS and 472 GFLOPS of Jetson Nano 4gb, and some loss of informations, blocks and operations during the model transformation from .PT to .ONNX to .ENGINE, the performance obtained worsening the CelsusSGM execution time and deserialize the model at each image , owing the Benchmark algorithm used force that, the average Time for each Middlebury image was 122.699867 [ms] and average of the average FPS is 27.44 [fps]. Another point to consider is the amount of parameters of StereoVae is bigger with the respect to well-known model as AnyNet or related. For the accuracy instead as all the Deep Learning methods need to use datasets with countless number of images (such as Scene Flow one) to enhance the results and reduce the errors. Despite that is noticeable the boost in the performance for our implementation of the Census SGM, and the map is then corrected by the StereoVAE reduce the noises and invalid

values and improve the disparities. The full advantage stands to have a model that works for images of all the sizes and disparity size and that can fastly post-processing the estimated output with a improved median filter and a fast quantized model.

IV. Conclusion

In this report we presented a efficient CUDA Census SGM , and a quantized StereoVAE to improve the final disparity. The limitations are similar as most of the Stereo Matching approaches plus having a worsening the accuracy in general and never reaching the state-of-art performances. This is due to the absence of a real convolution kernels. Despite this our approach is general and works with whatever image's size and disparity size. One possible improvement could be a training on a huge amount of data as CREStereo and Scene_Flow dataset to improve the accuracy and generalization, another improvement instead is substituting Census with ZNCC or supersede SGM with DT reducing the accuracy but increasing the number of FPS and reducing the execution time. There are several way to reduce the model in order to reduce the number of parameter and so reducing the execution time, e.g put an average pooling after Convolutional Layer and do Iterative Pruning. To draw a conclusion an efficient implementation of a traditional approach , CensusSGM, overcame a hybrid approach. This was something expected, for the execution time due the combination of both CensusSGM with the StereoVae model but with rooms of improvement to optimize both in terms of accuracy and to adapt to a real-time usage.

Acknowledgment

This work was supported by the whole 3D Data Processing class's staffs and sponsored by FlexSight Srl.

References

[1] GQ. Chang , X. Li , X. Xu, 'StereoVAE: A lightweight stereo-matching system using embedded GPUs" , May 2023 v1 and withdraw v2.

[2] Q. Chang, A. Zha, W. Wang, 'Efficient Stereo Matching on Embedded GPUs with Zero-Means Cross Correlation" , 1 Dec 2022.

[3] D. Hernandez-Juarez, A. Chacón, 'Embedded real-time stereo estimation via Semi-Global Matching on the GPU" , 2016.

[4] J. Zhao , T. Liang , L. Feng, 'FP-Stereo: Hardware-Efficient Stereo Vision for Embedded Applications" , 2020.

[5] H. Hirschmüller, 'Accurate and Efficient Stereo Processing by Semi-Global Matching and Mutual Information" , 2005.

[6] Y. Wang , Z. Lai , G. Huang, 'Anytime Stereo Image Depth Estimation on Mobile Devices" , 5 Mar 2019.

[7] Spangenberg, Robert and Langner, Tobias and Rojas, Raúl, ''Weighted Semi-Global Matching and Center-Symmetric Census Transform for Robust Driver Assistance', 2013.

V. Appendices

A. Build and Run Code

For all the C++ project just do :

SGM2016 :

- In the project folder ***mkdir build***
- ***cd build***
- ***cmake ..*** and then ***make***
- For execute just do:
  ./sgm
  -@left-image-format="$1" (required) format string for path to input left image
  -@right-image-format="$2" (required) format string for path to input right image
  -disparity_path="$3" (required) Path to store the PFM file
  -p1=7 (optional) SGM Penalization parameter $P1 < P2$
  -p2=84 (optional) SGM Smoothing parameter $P2 > P1$
- ***copy the executable to the Bash_Scripts/*** folder where there is the bash script
- the final test are available in the bash format to use them do :
  python3
  evaluate.py
  evaluate
  -d middlebury_quarter
  -a SGM2016
  -s sgm2016.sh
  -sl /WS/Principe_Sambin_NvidiaSM/
  SGM_EMB_2016/Bash_Scripts/

VPISGM :

- In the project folder ***mkdir build***
- ***cd build***
- ***cmake ..*** and then ***make***
- For execute just do:
  ./vpi_sample_02_stereo_disparity
  cuda (required) device to run the algorithm
  "$1" (required) format string for path to input left image
  "$2" (required) format string for path to input right image
  "$3" (required) Path to store the PFM file
- ***copy the executable to the Bash_Scripts/*** folder where there is the bash script
- the final test are available in the bash format to use them do :
  python3
  evaluate.py
  evaluate
  -d middlebury_quarter
  -a VPISGM
  -s sgm.sh
  -sl /WS/Principe_Sambin_NvidiaSM/
  VPI_SGM/Bash_Scripts/

CensusSGM :

- In the project folder ***mkdir build***
- ***cd build***
- ***cmake ..*** and then ***make***
- For execute just do:
  ./sgm_nano
  -@left-image-format="$1" (required) format string for path to input left image
  -@right-image-format="$2" (required) format string for path to input right image
  -disparity_path="$3" (required) Path to store the PFM

file
-p1=10 (optional) SGM Penalization parameter $P1 < P2$
-p2=120 (optional) SGM Smoothing parameter $P2 > P1$
-path_type=0 (optional) SGM Path it could be 4 = 0 or 8 = 1
-census_type=1 (optional) SGM Census type could be CENSUS_9x7 = 0 or SYMMETRIC_CENSUS_9x7 = 1
-sub_p="false" (optional) Sub Pixel precision to enhance the Disparity Map result
-reduce="true" (optional) reduce the size of the image before run Celsus SGM
-disp_size=64 (optional) maximum possible disparity value

- ***copy the executable to the Bash_Scripts/Final***, or in Bash_Scripts/Experiments/ or in any folder where there is the bash script
- all the ablation tests and the final test are available in the bash format to use them do :
  python3
  evaluate.py
  evaluate
  -d middlebury_quarter
  -a CensusSGM
  -s sgmcuda.sh or sgmcuda_f_64_4P_NoS.sh or other from the folder Bash_Scripts
  -sl /WS/Principe_Sambin_NvidiaSM/SGMCuda/Bash_Scripts/Final or /WS/Principe_Sambin_NvidiaSM/SGMCuda/Bash_Scripts/Experiments

StereoVae Project :

- In the project folder mkdir build
- ***cd build***
- ***cmake ..*** and then ***make***
- For test on only 1 pair of image, just do:
  ./sgm_nano
  -@left-image-format="$1" (required) format string for path to input left image
  -@right-image-format="$2" (required) format string for path to input right image
  -disparity_path="$3" (required) Path to store the PFM file
  -p1=10 (optional) SGM Penalization parameter $P1 < P2$
  -p2=120 (optional) SGM Smoothing parameter $P2 > P1$
  -path_type=0 (optional) SGM Path it could be 4 = 0 or 8 = 1
  -census_type=1 (optional) SGM Census type could be CENSUS_9x7 = 0 or SYMMETRIC_CENSUS_9x7 = 1
  -sub_p="false" (optional) Sub Pixel precision to enhance the Disparity Map result
  -reduce="true" (optional) reduce the size of the image before run Celsus SGM
  -disp_size=64 (optional) maximum possible disparity value
  -engine_or_onnx="/home/dp/WS/Principe_Sambin_Nvidia_Comp/StereoVae/stereovae_v17_mb20.engine" (required) if the aim is to create the engine this is the path to the ONNX model, otherwise is the Path of the Serialized Engine model
  -create=false (optional) mode Create if is wished to create the engine , is false if the engine is already created

- ***copy the executable to the Bash_Scripts/Final***, or in Bash_Scripts/Experiments/ or in any folder where there is the bash script and add the .engine file's address to the ***-engine_or_onnx*** in the bash file, for an istance look some lines above.
- the final test is available in the bash format to use them do :
  python3
  evaluate.py
  evaluate
  -d middlebury_quarter
  -a StereoVae
  -s stereovae.sh
  -sl /WS/Principe_Sambin_NvidiaSM/StereoVae/Bash_Scripts/

For the Jupyter Notebook is possible to use or Colab or Anaconda enviroment, the notebook is already provided with the link to the custom datasets, and instruction for uncommon libraries installation and path handling to store the model.

B. Disparity Maps from Middlebury

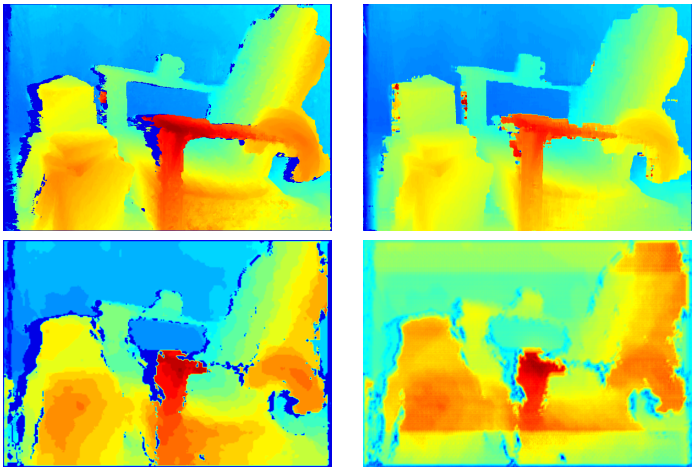

Fig. 4 Adirondack Middlebury Disparity Map :top left VPISGM outcome, top right SGM2016 outcome and bottom left CensusSGM with parameters [DISPARITY SIZE = 64, P1=10, P2=120, SUBPIX=FALSE, 4 PATHS, SYMMETRIC CENSUS] and bottom right the StereoVae outcome.

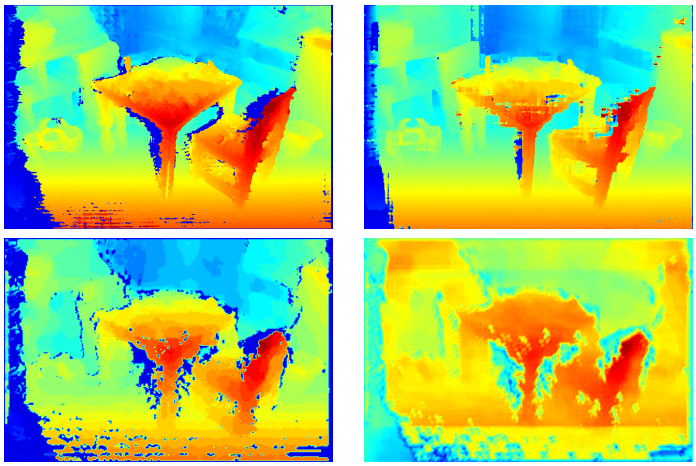

Fig. 5 Playtable Middlebury Disparity Map :top left VPISGM outcome, top right SGM2016 outcome and bottom left CensusSGM with parameters [DISPARITY SIZE = 64, P1=10, P2=120, SUBPIX=FALSE, 4 PATHS, SYMMETRIC CENSUS] and bottom right the StereoVae outcome..

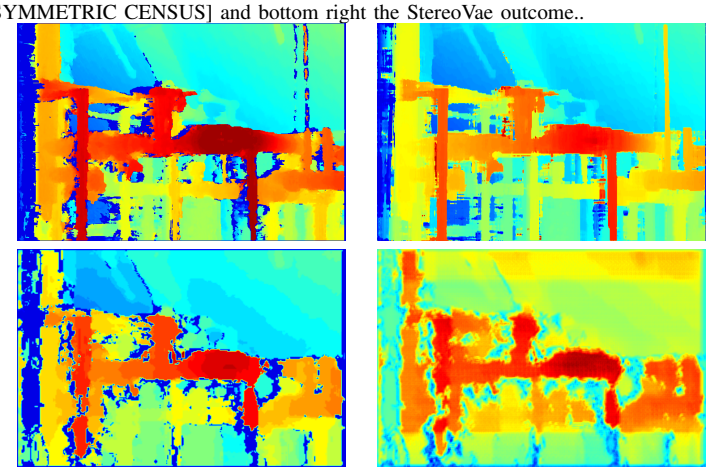

Fig. 6 Pipes Middlebury Disparity Map :top left VPISGM outcome, top right SGM2016 outcome and bottom left CensusSGM with parameters [DISPARITY SIZE = 64, P1=10, P2=120, SUBPIX=FALSE, 4 PATHS, SYMMETRIC CENSUS] and bottom right the StereoVae outcome.

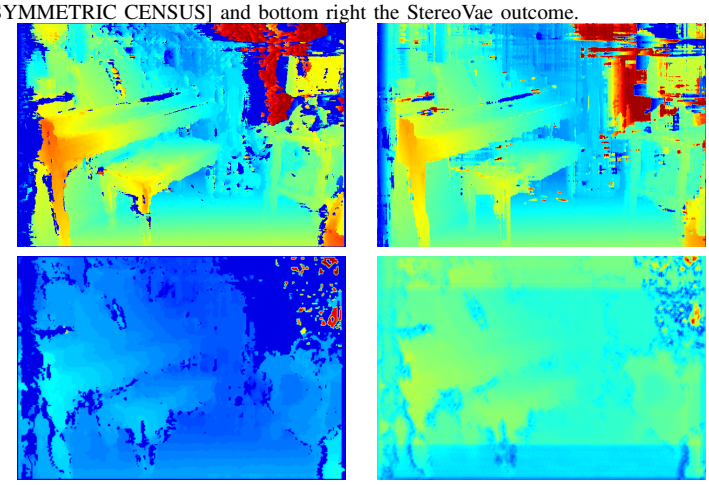

Fig. 7 PianoL Middlebury Disparity Map : top left VPISGM outcome, top right SGM2016 outcome and bottom left CensusSGM with parameters [DISPARITY SIZE = 64, P1=10, P2=120, SUBPIX=FALSE, 4 PATHS, SYMMETRIC CENSUS] and bottom right the StereoVae outcome.

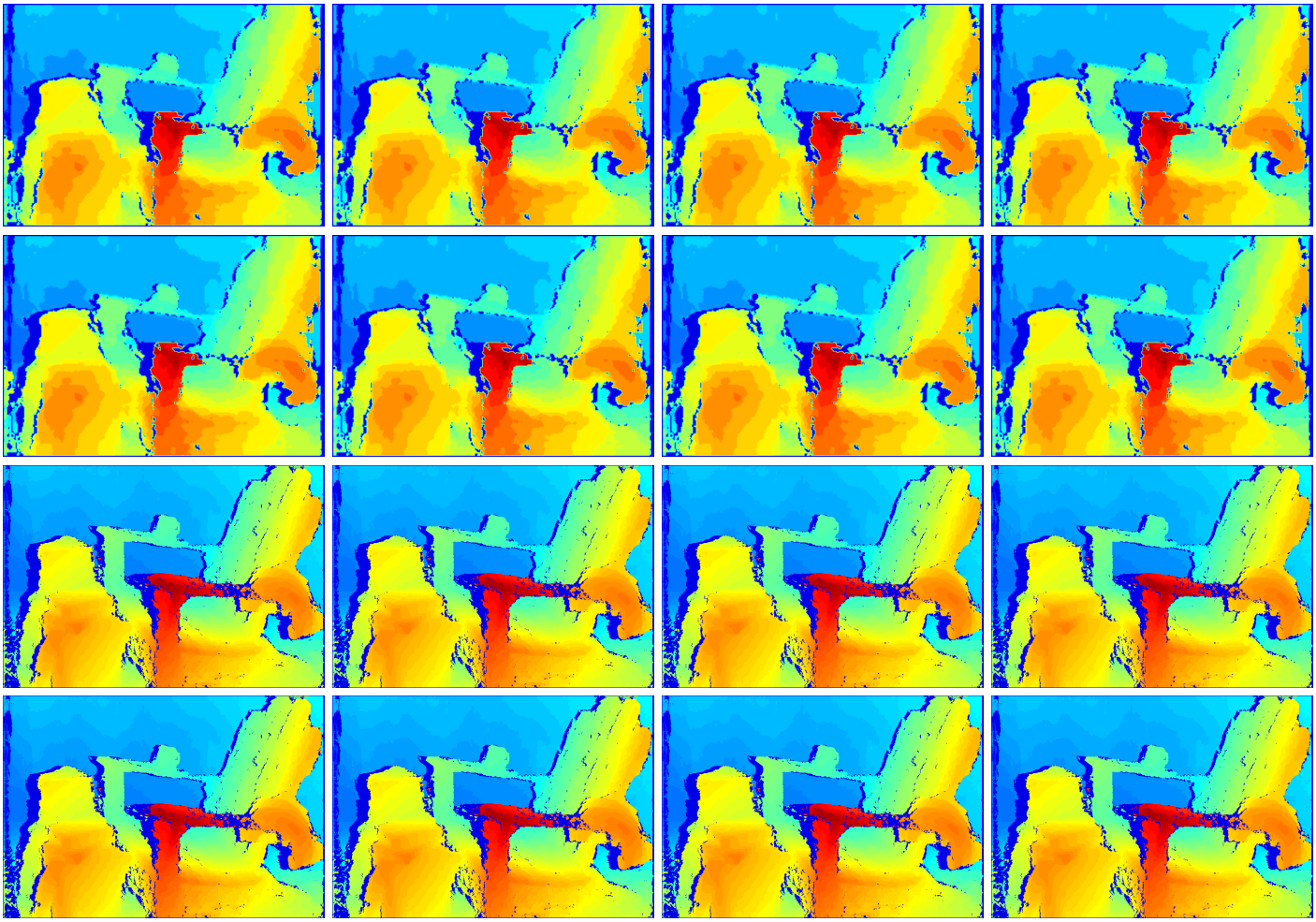

Fig. 4. Fig. 8 Adirondack Middlebury Disparity Map : 1st row CensusSGM Reduced with parameter [Disparity Size = 64], 2nd Row CensusSGM Reduced with parameter [Disparity Size = 128], 3rd Row CensusSGM Full with parameter [Disparity Size = 128] and 4rth Row CensusSGM Full with parameters [DISPARITY SIZE = 64]. 1st Column [Paths = 4, No SubPixel Accuracy], 2nd Column [Paths = 4, Subpixel Accuracy], 3rd Column [Paths = 8, No Subpixel Accuracy] and 4rth Column [Paths = 8, Subpixel Accuracy], all the models has parameters [Symmetric Census, P1=10, P2=120]

C. Full Algorithms' Benchmarks Ablation Study

dataset	algorithm	bad_0_5	bad_1_0	bad_2_0	bad_4_0	t_bad_0_5	t_bad_1_0	t_bad_2_0	t_bad_4_0	avgerr	invalid
Adirondack	CensusSGM	90.8738	82.0724	65.0866	37.6659	90.8738	82.0724	65.0866	37.6659	12.736	0
Adirondack	**SGM2016**	70.282	**45.6145**	**22.4447**	**9.97353**	70.282	**45.6145**	**22.4447**	**9.97353**	**3.45081**	0
Adirondack	VPISGM	**69.7065**	46.1292	24.2556	12.0706	**69.7065**	46.1292	24.2556	12.0706	4.59509	0
Adirondack	StereoVae	99.6398	99.2697	98.5585	97.0488	99.6398	99.2697	98.5585	97.0488	36.9421	0
ArtL	CensusSGM	83.6552	59.6468	45.0365	35.5548	83.6552	59.6468	45.0365	35.5548	24.8037	0
ArtL	SGM2016	79.2354	62.9962	57.6924	52.998	79.2354	62.9962	57.6924	52.998	37.6083	0
ArtL	**VPISGM**	**76.5438**	**43.8121**	**32.3326**	**24.6256**	**76.5438**	**43.8121**	**32.3326**	**24.6256**	**13.0396**	0
ArtL	StereoVae	99.3087	98.0636	96.9255	94.5235	99.3087	98.0636	96.9255	94.5235	50.8251	0
Jadeplant	CensusSGM	94.0287	87.4788	74.6021	56.7732	94.0287	87.4788	74.6021	**56.7732**	**88.8222**	0
Jadeplant	**SGM2016**	**87.6844**	**77.8821**	**70.1394**	**66.3659**	**87.6844**	**77.8821**	**70.1394**	66.3659	136.035	0
Jadeplant	VPISGM	89.6193	81.94	75.1657	71.4866	89.6193	81.94	75.1657	71.4866	172.7	0
Jadeplant	StereoVae	99.9762	99.9542	99.9108	99.8233	99.9762	99.9542	99.9108	99.8233	169.341	0
Motorcycle	CensusSGM	91.458	83.1546	66.4322	36.9921	91.458	83.1546	66.4322	36.9921	13.4302	0
Motorcycle	**SGM2016**	**69.6252**	**45.4014**	**22.1076**	**11.0948**	**69.6252**	**45.4014**	**22.1076**	**11.0948**	**4.39406**	0
Motorcycle	VPISGM	70.4367	47.3917	23.9314	12.8995	70.4367	47.3917	23.9314	12.8995	6.19043	0
Motorcycle	StereoVae	98.9936	97.9837	96.0371	92.3276	98.9936	97.9837	96.0371	92.3276	39.6919	0
MotorcycleE	CensusSGM	91.4346	83.1027	66.3713	36.8648	91.4346	83.1027	66.3713	36.8648	13.3965	0
MotorcycleE	**SGM2016**	68.9958	44.8661	**21.232**	**10.7326**	68.9958	44.8661	**21.232**	**10.7326**	**4.30207**	0
MotorcycleE	VPISGM	**68.2774**	**44.4423**	22.2787	12.191	**68.2774**	**44.4423**	22.2787	12.191	6.03382	0
MotorcycleE	StereoVae	98.8237	97.6457	95.107	89.9063	98.8237	97.6457	95.107	89.9063	37.7958	0
Piano	CensusSGM	92.8361	85.5996	71.0218	47.018	92.8361	85.5996	71.0218	47.018	12.0728	0
Piano	**SGM2016**	71.4783	49.0069	**28.2035**	**18.712**	71.4783	49.0069	**28.2035**	**18.712**	**3.73542**	0
Piano	VPISGM	**69.6748**	**46.9965**	28.2448	19.3944	**69.6748**	**46.9965**	28.2448	19.3944	5.67917	0
Piano	StereoVae	99.9237	99.8487	99.6892	99.3408	99.9237	99.8487	99.6892	99.3408	64.2563	0
PianoL	CensusSGM	94.364	88.6211	77.2467	59.1385	94.364	88.6211	77.2467	59.1385	28.5255	0
PianoL	**SGM2016**	81.0965	65.6839	**48.5987**	**38.2684**	81.0965	65.6839	**48.5987**	**38.2684**	**20.74**	0
PianoL	VPISGM	**80.0945**	**64.6833**	49.5318	39.9326	**80.0945**	**64.6833**	49.5318	39.9326	23.4377	0
PianoL	StereoVae	98.9988	98.0211	96.2063	92.5112	98.9988	98.0211	96.2063	92.5112	28.4947	0
Pipes	CensusSGM	92.0041	84.2578	70.4309	46.6253	92.0041	84.2578	70.4309	46.6253	27.3098	0
Pipes	**SGM2016**	69.0004	**43.5559**	**23.4793**	**16.1127**	69.0004	**43.5559**	**23.4793**	**16.1127**	**8.23626**	0
Pipes	VPISGM	**68.9035**	45.1314	28.0684	20.7715	**68.9035**	45.1314	28.0684	20.7715	12.284	0
Pipes	StereoVae	99.187	98.3545	96.7184	93.5367	99.187	98.3545	96.7184	93.5367	44.1085	0
Playroom	CensusSGM	93.2715	86.4985	73.1516	51.2103	93.2715	86.4985	73.1516	51.2103	24.5012	0
Playroom	**SGM2016**	**77.836**	**59.3768**	**37.7878**	**23.3421**	**77.836**	**59.3768**	**37.7878**	**23.3421**	**6.68388**	0
Playroom	VPISGM	78.9056	61.8468	42.6384	30.2981	78.9056	61.8468	42.6384	30.2981	14.3854	0
Playroom	StereoVae	99.4303	98.8569	97.8256	95.8588	99.4303	98.8569	97.8256	95.8588	50.1008	0
Playtable	CensusSGM	92.6802	85.3543	71.3607	49.1427	92.6802	85.3543	71.3607	49.1427	21.5731	0
Playtable	**SGM2016**	**76.6067**	**56.4696**	**33.5893**	**21.0722**	**76.6067**	**56.4696**	**33.5893**	**21.0722**	**5.82278**	0
Playtable	VPISGM	78.8638	59.0955	36.0272	22.6348	78.8638	59.0955	36.0272	22.6348	10.7107	0
Playtable	StereoVae	99.4673	98.9354	97.7988	95.5369	99.4673	98.9354	97.7988	95.5369	40.9254	0
PlaytableP	CensusSGM	92.367	84.8133	69.9924	47.7498	92.367	84.8133	69.9924	47.7498	20.5748	0
PlaytableP	SGM2016	67.3285	44.9791	28.6457	18.6168	67.3285	44.9791	28.6457	18.6168	**4.80113**	0
PlaytableP	**VPISGM**	**65.6333**	**43.2891**	**27.3434**	**17.9133**	**65.6333**	**43.2891**	**27.3434**	**17.9133**	6.99493	0
PlaytableP	StereoVae	99.375	98.7555	97.5078	95.1464	99.375	98.7555	97.5078	95.1464	38.5265	0
Recycle	CensusSGM	92.0839	83.8659	67.7598	39.0567	92.0839	83.8659	67.7598	39.0567	10.9219	0
Recycle	**SGM2016**	69.7443	**45.4374**	**23.8702**	**13.2162**	69.7443	**45.4374**	**23.8702**	**13.2162**	**3.51232**	0
Recycle	VPISGM	**69.7383**	46.9733	26.5835	15.8142	**69.7383**	46.9733	26.5835	15.8142	4.97983	0
Recycle	StereoVae	99.5432	99.0851	98.1789	96.2785	99.5432	99.0851	98.1789	96.2785	33.7444	0
Shelves	CensusSGM	94.829	89.7514	80.0662	60.6791	94.829	89.7514	80.0662	60.6791	16.8095	0
Shelves	**SGM2016**	**83.6426**	**69.3892**	**53.3761**	43.0968	**83.6426**	**69.3892**	**53.3761**	43.0968	**10.8084**	0
Shelves	VPISGM	84.1016	70.7447	54.5115	**41.8838**	84.1016	70.7447	54.5115	**41.8838**	10.8685	0
Shelves	StereoVae	99.9029	99.8024	99.5865	99.1134	99.9029	99.8024	99.5865	99.1134	68.9153	0
Teddy	CensusSGM	87.5642	59.1562	36.8711	20.882	87.5642	59.1562	36.8711	20.882	10.4977	0

Teddy	**SGM2016**	**63.9393**	**21.8258**	**13.1358**	**9.14958**	**63.9393**	**21.8258**	**13.1358**	**9.14958**	**3.27023**	0
Teddy	VPISGM	67.8861	25.6392	15.6046	11.4717	67.8861	25.6392	15.6046	11.4717	5.21508	0
Teddy	StereoVae	99.9077	99.7274	99.5517	99.2032	99.9077	99.7274	99.5517	99.2032	71.9422	0
Vintage	CensusSGM	93.9136	87.9511	76.0132	55.3247	93.9136	87.9511	76.0132	55.3247	**37.3693**	0
Vintage	**SGM2016**	**84.1177**	**71.7088**	**59.8325**	**52.9985**	**84.1177**	**71.7088**	**59.8325**	**52.9985**	58.6143	0
Vintage	VPISGM	89.6006	81.2363	72.1384	65.585	89.6006	81.2363	72.1384	65.585	92.7691	0
Vintage	StereoVae	99.3975	98.7787	97.5151	95.093	99.3975	98.7787	97.5151	95.093	89.1483	0

TABLE III: Statistics Benchmarks with the Middlebury Dataset of all the SM Algorithm referenced in this report : from [3] SGM_2016 with parameter [Size Disparity = 128, size of the image = Kitti Image size, P1=7 , P2=84] , VPISGM (Nvidia VPI library version) and our CensusSGM with parameter [Reduced image, P1=10, P2=120, SubPix=False, Size Disparity = 64, 4 Paths, Symmetric CENSUS]

dataset	algorithm	Time	FPS
Adirondack	**CensusSGM**	**101.787 [ms]**	**9.82444 [fps]**
Adirondack	SGM2016	152.968 [ms]	6.53731 [fps]
Adirondack	VPISGM	275.546082 [ms]	3.629157 [fps]
Adirondack	StereoVae	135.196 [ms]	20.2485 [fps]
ArtL	**CensusSGM**	**60.357 [ms]**	**16.5681 [fps]**
ArtL	SGM2016	78.196 [ms]	12.7884 [fps]
ArtL	VPISGM	90.708336 [ms]	11.024345 [fps]
ArtL	StereoVae	132.575 [ms]	21.1635 [fps]
Jadeplant	**CensusSGM**	**60.6 [ms]**	**16.5017 [fps]**
Jadeplant	SGM2016	98.7391 [ms]	10.1277 [fps]
Jadeplant	VPISGM	77.984589 [ms]	12.823047 [fps]
Jadeplant	StereoVae	149.196 [ms]	22.9566 [fps]
Motorcycle	**CensusSGM**	**46.602 [ms]**	**21.4583 [fps]**
Motorcycle	SGM2016	83.1618 [ms]	12.0247 [fps]
Motorcycle	VPISGM	72.404793 [ms]	13.811240 [fps]
Motorcycle	StereoVae	127.016 [ms]	24.6953 [fps]
MotorcycleE	**CensusSGM**	**46.521 [ms]**	**21.4957 [fps]**
MotorcycleE	SGM2016	83.5102 [ms]	11.9746 [fps]
MotorcycleE	VPISGM	64.718079 [ms]	15.451633 [fps]
MotorcycleE	StereoVae	105.816 [ms]	25.0279 [fps]
Piano	**CensusSGM**	**46.644 [ms]**	**21.439 [fps]**
Piano	SGM2016	84.9858 [ms]	11.7667 [fps]
Piano	VPISGM	60.524067 [ms]	16.522352 [fps]
Piano	StereoVae	101.599 [ms]	25.7337 [fps]
PianoL	**CensusSGM**	**46.433 [ms]**	**21.5364 [fps]**
PianoL	SGM2016	80.3106 [ms]	12.4517 [fps]
PianoL	VPISGM	63.437611 [ms]	15.763519 [fps]
PianoL	StereoVae	122.685 [ms]	28.7195 [fps]
Pipes	CensusSGM	47.007 [ms]	21.2734 [fps]
Pipes	SGM2016	87.3505 [ms]	11.4481 [fps]
Pipes	**VPISGM**	**33.546822 [ms]**	**29.809084 [fps]**
Pipes	StereoVae	119.473 [ms]	32.644 [fps]
Playroom	**CensusSGM**	**46.762 [ms]**	**21.3849 [fps]**
Playroom	SGM2016	82.1369 [ms]	12.1748 [fps]
Playroom	VPISGM	62.610939 [ms]	15.971650 [fps]
Playroom	StereoVae	123.55 [ms]	28.2627 [fps]
Playtable	**CensusSGM**	**46.905 [ms]**	**21.3197 [fps]**
Playtable	SGM2016	82.0591 [ms]	12.1863 [fps]
Playtable	VPISGM	58.671402 [ms]	17.044079 [fps]
Playtable	StereoVae	124.591 [ms]	26.6276 [fps]
PlaytableP	**CensusSGM**	**47.285 [ms]**	**21.1484 [fps]**
PlaytableP	SGM2016	82.0722 [ms]	12.1844 [fps]
PlaytableP	VPISGM	67.854218 [ms]	14.737477 [fps]
PlaytableP	StereoVae	122.761 [ms]	28.593 [fps]
Recycle	**CensusSGM**	**47.243 [ms]**	**21.1672 [fps]**
Recycle	SGM2016	81.7466 [ms]	12.2329 [fps]
Recycle	VPISGM	60.757446 [ms]	16.458887 [fps]
Recycle	StereoVae	115.271 [ms]	31.187 [fps]
Shelves	**CensusSGM**	**46.92 [ms]**	**21.3129 [fps]**
Shelves	SGM2016	82.7447 [ms]	12.0854 [fps]
Shelves	VPISGM	69.093597 [ms]	14.473121 [fps]
Shelves	StereoVae	120.511 [ms]	31.2266 [fps]
Teddy	CensusSGM	46.989 [fps]	21.2816 [fps]

Teddy	SGM2016	82.2905 [ms]	12.1521 [fps]
Teddy	**VPISGM**	**18.643961 [ms]**	**53.636669 [fps]**
Teddy	StereoVae	121.642 [ms]	30.1834 [fps]
Vintage	**CensusSGM**	**39.959 [ms]**	**25.0257 [fps]**
Vintage	SGM2016	82.7357 [ms]	12.0867 [fps]
Vintage	VPISGM	63.204899 [ms]	15.821558 [fps]
Vintage	StereoVae	118.616 [ms]	34.3307 [fps]

TABLE IV: Time of Execution in milliseconds (ms) and FPS Benchmarks with the Middlebury Dataset of all the SM Algorithm referenced in this report : from [3] SGM_2016, VPISGM (Nvidia VPI library version) and our CensusSGM with parameter [Reduced image, P1=10, P2=120, SubPix=False, Size Disparity = 64, 4 Paths, Symmetric CENSUS]

dataset	algorithm	bad_0_5	bad_1_0	bad_2_0	bad_4_0	t_bad_0_5	t_bad_1_0	t_bad_2_0	t_bad_4_0	avgerr
Adirondack	**CensusSGM_f_128_4P_NoS**	**80.1088**	**61.0256**	**30.6703**	**13.3178**	**80.1088**	**61.0256**	**30.6703**	**13.3178**	8.64672
Adirondack	CensusSGM_f_128_4P_S	80.1088	61.0256	30.6703	13.3178	80.1088	61.0256	30.6703	13.3178	8.64672
Adirondack	CensusSGM_f_128_8P_NoS	80.1088	61.0256	30.6703	13.3178	80.1088	61.0256	30.6703	13.3178	8.64672
Adirondack	CensusSGM_f_128_8P_S	80.1088	61.0256	30.6703	13.3178	80.1088	61.0256	30.6703	13.3178	8.64672
Adirondack	**CensusSGM_f_64_4P_NoS**	80.1132	61.0351	30.6773	13.3148	80.1132	61.0351	30.6773	13.3148	**8.62903**
Adirondack	CensusSGM_f_64_4P_S	80.1132	61.0351	30.6773	13.3148	80.1132	61.0351	30.6773	13.3148	8.62903
Adirondack	CensusSGM_f_64_8P_NoS	80.1132	61.0351	30.6773	13.3148	80.1132	61.0351	30.6773	13.3148	8.62903
Adirondack	CensusSGM_f_64_8P_S	80.1132	61.0351	30.6773	13.3148	80.1132	61.0351	30.6773	13.3148	8.62903
Adirondack	CensusSGM_r_128_4P_NoS	90.8737	82.0735	65.092	37.6689	90.8737	82.0735	65.092	37.6689	12.7434
Adirondack	CensusSGM_r_128_4P_S	90.8737	82.0735	65.092	37.6689	90.8737	82.0735	65.092	37.6689	12.7434
Adirondack	CensusSGM_r_128_8P_NoS	90.8737	82.0735	65.092	37.6689	90.8737	82.0735	65.092	37.6689	12.7434
Adirondack	CensusSGM_r_128_8P_S	90.8737	82.0735	65.092	37.6689	90.8737	82.0735	65.092	37.6689	12.7434
Adirondack	CensusSGM_r_64_4P_NoS_C0	90.8738	82.0724	65.0866	37.6659	90.8738	82.0724	65.0866	37.6659	12.736
Adirondack	CensusSGM_r_64_4P_NoS	90.8738	82.0724	65.0866	37.6659	90.8738	82.0724	65.0866	37.6659	12.736
Adirondack	CensusSGM_r_64_4P_S	90.8738	82.0724	65.0866	37.6659	90.8738	82.0724	65.0866	37.6659	12.736
Adirondack	CensusSGM_r_64_8P_NoS	90.8738	82.0724	65.0866	37.6659	90.8738	82.0724	65.0866	37.6659	12.736
Adirondack	CensusSGM_r_64_8P_S	90.8738	82.0724	65.0866	37.6659	90.8738	82.0724	65.0866	37.6659	12.736
ArtL	CensusSGM_f_128_4P_NoS	81.5368	49.4942	34.6693	27.5716	81.5368	49.4942	34.6693	27.5716	22.037
ArtL	CensusSGM_f_128_4P_S	81.5368	49.4942	34.6693	27.5716	81.5368	49.4942	34.6693	27.5716	22.037
ArtL	CensusSGM_f_128_8P_NoS	81.5368	49.4942	34.6693	27.5716	81.5368	49.4942	34.6693	27.5716	22.037
ArtL	CensusSGM_f_128_8P_S	81.5368	49.4942	34.6693	27.5716	81.5368	49.4942	34.6693	27.5716	22.037
ArtL	**CensusSGM_f_64_4P_NoS**	**81.528**	**49.4838**	**34.6516**	**27.5658**	**81.528**	**49.4838**	**34.6516**	**27.5658**	**21.9538**
ArtL	CensusSGM_f_64_4P_S	81.528	49.4838	34.6516	27.5658	81.528	49.4838	34.6516	27.5658	21.9538
ArtL	CensusSGM_f_64_8P_NoS	81.528	49.4838	34.6516	27.5658	81.528	49.4838	34.6516	27.5658	21.9538
ArtL	CensusSGM_f_64_8P_S	81.528	49.4838	34.6516	27.5658	81.528	49.4838	34.6516	27.5658	21.9538
ArtL	CensusSGM_r_128_4P_NoS	83.6242	59.6093	44.9557	35.4495	83.6242	59.6093	44.9557	35.4495	24.6227
ArtL	CensusSGM_r_128_4P_S	83.6242	59.6093	44.9557	35.4495	83.6242	59.6093	44.9557	35.4495	24.6227
ArtL	CensusSGM_r_128_8P_NoS	83.6242	59.6093	44.9557	35.4495	83.6242	59.6093	44.9557	35.4495	24.6227
ArtL	CensusSGM_r_128_8P_S	83.6242	59.6093	44.9557	35.4495	83.6242	59.6093	44.9557	35.4495	24.6227
ArtL	CensusSGM_r_64_4P_NoS_C0	83.6552	59.6468	45.0365	35.5548	83.6552	59.6468	45.0365	35.5548	24.8037
ArtL	CensusSGM_r_64_4P_NoS	83.6552	59.6468	45.0365	35.5548	83.6552	59.6468	45.0365	35.5548	24.8037
ArtL	CensusSGM_r_64_4P_S	83.6552	59.6468	45.0365	35.5548	83.6552	59.6468	45.0365	35.5548	24.8037
ArtL	CensusSGM_r_64_8P_NoS	83.6552	59.6468	45.0365	35.5548	83.6552	59.6468	45.0365	35.5548	24.8037
ArtL	CensusSGM_r_64_8P_S	83.6552	59.6468	45.0365	35.5548	83.6552	59.6468	45.0365	35.5548	24.8037
Jadeplant	**CensusSGM_f_128_4P_NoS**	**84.7528**	**70.241**	**49.2591**	**38.7522**	**84.7528**	**70.241**	**49.2591**	**38.7522**	**89.9764**
Jadeplant	CensusSGM_f_128_4P_S	84.7528	70.241	49.2591	38.7522	84.7528	70.241	49.2591	38.7522	89.9764
Jadeplant	CensusSGM_f_128_8P_NoS	84.7528	70.241	49.2591	38.7522	84.7528	70.241	49.2591	38.7522	89.9764
Jadeplant	CensusSGM_f_128_8P_S	84.7528	70.241	49.2591	38.7522	84.7528	70.241	49.2591	38.7522	89.9764
Jadeplant	CensusSGM_f_64_4P_NoS	92.3889	85.0969	75.3391	70.9346	92.3889	85.0969	75.3391	70.9346	188.779
Jadeplant	CensusSGM_f_64_4P_S	92.3889	85.0969	75.3391	70.9346	92.3889	85.0969	75.3391	70.9346	188.779
Jadeplant	CensusSGM_f_64_8P_NoS	92.3889	85.0969	75.3391	70.9346	92.3889	85.0969	75.3391	70.9346	188.779
Jadeplant	CensusSGM_f_64_8P_S	92.3889	85.0969	75.3391	70.9346	92.3889	85.0969	75.3391	70.9346	188.779
Jadeplant	CensusSGM_r_128_4P_NoS	93.862	87.1623	74.0289	55.9151	93.862	87.1623	74.0289	55.9151	88.1005
Jadeplant	CensusSGM_r_128_4P_S	93.862	87.1623	74.0289	55.9151	93.862	87.1623	74.0289	55.9151	88.1005
Jadeplant	CensusSGM_r_128_8P_NoS	93.862	87.1623	74.0289	55.9151	93.862	87.1623	74.0289	55.9151	88.1005
Jadeplant	CensusSGM_r_128_8P_S	93.862	87.1623	74.0289	55.9151	93.862	87.1623	74.0289	55.9151	88.1005
Jadeplant	CensusSGM_r_64_4P_NoS_C0	94.0287	87.4788	74.6021	56.7732	94.0287	87.4788	74.6021	56.7732	88.8222
Jadeplant	CensusSGM_r_64_4P_NoS	94.0287	87.4788	74.6021	56.7732	94.0287	87.4788	74.6021	56.7732	88.8222
Jadeplant	CensusSGM_r_64_4P_S	94.0287	87.4788	74.6021	56.7732	94.0287	87.4788	74.6021	56.7732	88.8222
Jadeplant	CensusSGM_r_64_8P_NoS	94.0287	87.4788	74.6021	56.7732	94.0287	87.4788	74.6021	56.7732	88.8222
Jadeplant	CensusSGM_r_64_8P_S	94.0287	87.4788	74.6021	56.7732	94.0287	87.4788	74.6021	56.7732	88.8222
Motorcycle	**CensusSGM_f_128_4P_NoS**	**80.0498**	**61.352**	**31.6191**	**14.1372**	**80.0498**	**61.352**	**31.6191**	**14.1372**	**9.07868**
Motorcycle	CensusSGM_f_128_4P_S	80.0498	61.352	31.6191	14.1372	80.0498	61.352	31.6191	14.1372	9.07868
Motorcycle	CensusSGM_f_128_8P_NoS	80.0498	61.352	31.6191	14.1372	80.0498	61.352	31.6191	14.1372	9.07868

Motorcycle	CensusSGM_f_128_8P_S	80.0498	61.352	31.6191	14.1372	80.0498	61.352	31.6191	14.1372	9.07868
Motorcycle	CensusSGM_f_64_4P_NoS	80.052	61.3554	31.6236	14.1415	80.052	61.3554	31.6236	14.1415	9.06394
Motorcycle	CensusSGM_f_64_4P_S	80.052	61.3554	31.6236	14.1415	80.052	61.3554	31.6236	14.1415	9.06394
Motorcycle	CensusSGM_f_64_8P_NoS	80.052	61.3554	31.6236	14.1415	80.052	61.3554	31.6236	14.1415	9.06394
Motorcycle	CensusSGM_f_64_8P_S	80.052	61.3554	31.6236	14.1415	80.052	61.3554	31.6236	14.1415	9.06394
Motorcycle	CensusSGM_r_128_4P_NoS	91.4534	83.1555	66.4339	36.9945	91.4534	83.1555	66.4339	36.9945	13.4286
Motorcycle	CensusSGM_r_128_4P_S	91.4534	83.1555	66.4339	36.9945	91.4534	83.1555	66.4339	36.9945	13.4286
Motorcycle	CensusSGM_r_128_8P_NoS	91.4534	83.1555	66.4339	36.9945	91.4534	83.1555	66.4339	36.9945	13.4286
Motorcycle	CensusSGM_r_128_8P_S	91.4534	83.1555	66.4339	36.9945	91.4534	83.1555	66.4339	36.9945	13.4286
Motorcycle	CensusSGM_r_64_4P_NoS_C0	91.458	83.1546	66.4322	36.9921	91.458	83.1546	66.4322	36.9921	13.4302
Motorcycle	CensusSGM_r_64_4P_NoS	91.458	83.1546	66.4322	36.9921	91.458	83.1546	66.4322	36.9921	13.4302
Motorcycle	CensusSGM_r_64_4P_S	91.458	83.1546	66.4322	36.9921	91.458	83.1546	66.4322	36.9921	13.4302
Motorcycle	CensusSGM_r_64_8P_NoS	91.458	83.1546	66.4322	36.9921	91.458	83.1546	66.4322	36.9921	13.4302
Motorcycle	CensusSGM_r_64_8P_S	91.458	83.1546	66.4322	36.9921	91.458	83.1546	66.4322	36.9921	13.4302
MotorcycleE	**CensusSGM_f_128_P_NoS**	**79.8095**	**60.8093**	**30.7867**	13.6509	**79.8095**	**60.8093**	**30.7867**	13.6509	8.97663
MotorcycleE	CensusSGM_f_128_4P_S	79.8095	60.8093	30.7867	13.6509	79.8095	60.8093	30.7867	13.6509	8.97663
MotorcycleE	CensusSGM_f_128_8P_NoS	79.8095	60.8093	30.7867	13.6509	79.8095	60.8093	30.7867	13.6509	8.97663
MotorcycleE	CensusSGM_f_128_8P_S	79.8095	60.8093	30.7867	13.6509	79.8095	60.8093	30.7867	13.6509	8.97663
MotorcycleE	CensusSGM_f_64_4P_NoS	79.8125	60.8126	30.7915	**13.6498**	79.8125	60.8126	30.7915	**13.6498**	**8.94759**
MotorcycleE	CensusSGM_f_64_4P_S	79.8125	60.8126	30.7915	13.6498	79.8125	60.8126	30.7915	13.6498	8.94759
MotorcycleE	CensusSGM_f_64_8P_NoS	79.8125	60.8126	30.7915	13.6498	79.8125	60.8126	30.7915	13.6498	8.94759
MotorcycleE	CensusSGM_f_64_8P_S	79.8125	60.8126	30.7915	13.6498	79.8125	60.8126	30.7915	13.6498	8.94759
MotorcycleE	CensusSGM_r_128_4P_NoS	91.426	83.0955	66.365	36.8548	91.426	83.0955	66.365	36.8548	13.3761
MotorcycleE	CensusSGM_r_128_4P_S	91.426	83.0955	66.365	36.8548	91.426	83.0955	66.365	36.8548	13.3761
MotorcycleE	CensusSGM_r_128_8P_NoS	91.426	83.0955	66.365	36.8548	91.426	83.0955	66.365	36.8548	13.3761
MotorcycleE	CensusSGM_r_128_8P_S	91.426	83.0955	66.365	36.8548	91.426	83.0955	66.365	36.8548	13.3761
MotorcycleE	CensusSGM_r_64_4P_NoS_C0	91.4346	83.1027	66.3713	36.8648	91.4346	83.1027	66.3713	36.8648	13.3965
MotorcycleE	CensusSGM_r_64_4P_NoS	91.4346	83.1027	66.3713	36.8648	91.4346	83.1027	66.3713	36.8648	13.3965
MotorcycleE	CensusSGM_r_64_4P_S	91.4346	83.1027	66.3713	36.8648	91.4346	83.1027	66.3713	36.8648	13.3965
MotorcycleE	CensusSGM_r_64_8P_NoS	91.4346	83.1027	66.3713	36.8648	91.4346	83.1027	66.3713	36.8648	13.3965
MotorcycleE	CensusSGM_r_64_8P_S	91.4346	83.1027	66.3713	36.8648	91.4346	83.1027	66.3713	36.8648	13.3965
Piano	**CensusSGM_f_128_4P_NoS**	**83.1277**	**66.6288**	**40.198**	**25.175**	**83.1277**	**66.6288**	**40.198**	**25.175**	**12.2692**
Piano	CensusSGM_f_128_4P_S	83.1277	66.6288	40.198	25.175	83.1277	66.6288	40.198	25.175	12.2692
Piano	CensusSGM_f_128_8P_NoS	83.1277	66.6288	40.198	25.175	83.1277	66.6288	40.198	25.175	12.2692
Piano	CensusSGM_f_128_8P_S	83.1277	66.6288	40.198	25.175	83.1277	66.6288	40.198	25.175	12.2692
Piano	CensusSGM_f_64_4P_NoS	83.1282	66.6299	40.1997	25.1797	83.1282	66.6299	40.1997	25.1797	12.2697
Piano	CensusSGM_f_64_4P_S	83.1282	66.6299	40.1997	25.1797	83.1282	66.6299	40.1997	25.1797	12.2697
Piano	CensusSGM_f_64_8P_NoS	83.1282	66.6299	40.1997	25.1797	83.1282	66.6299	40.1997	25.1797	12.2697
Piano	CensusSGM_f_64_8P_S	83.1282	66.6299	40.1997	25.1797	83.1282	66.6299	40.1997	25.1797	12.2697
Piano	CensusSGM_r_128_4P_NoS	92.8364	85.5977	71.017	47.0146	92.8364	85.5977	71.017	47.0146	12.0655
Piano	CensusSGM_r_128_4P_S	92.8364	85.5977	71.017	47.0146	92.8364	85.5977	71.017	47.0146	12.0655
Piano	CensusSGM_r_128_8P_NoS	92.8364	85.5977	71.017	47.0146	92.8364	85.5977	71.017	47.0146	12.0655
Piano	CensusSGM_r_128_8P_S	92.8364	85.5977	71.017	47.0146	92.8364	85.5977	71.017	47.0146	12.0655
Piano	CensusSGM_r_64_4P_NoS_C0	92.8361	85.5996	71.0218	47.018	92.8361	85.5996	71.0218	47.018	12.0728
Piano	CensusSGM_r_64_4P_NoS	92.8361	85.5996	71.0218	47.018	92.8361	85.5996	71.0218	47.018	12.0728
Piano	CensusSGM_r_64_4P_S	92.8361	85.5996	71.0218	47.018	92.8361	85.5996	71.0218	47.018	12.0728
Piano	CensusSGM_r_64_8P_NoS	92.8361	85.5996	71.0218	47.018	92.8361	85.5996	71.0218	47.018	12.0728
Piano	CensusSGM_r_64_8P_S	92.8361	85.5996	71.0218	47.018	92.8361	85.5996	71.0218	47.018	12.0728
PianoL	CensusSGM_f_128_4P_NoS	86.5567	73.4471	52.6564	38.7157	86.5567	73.4471	52.6564	38.7157	**26.8717**
PianoL	CensusSGM_f_128_4P_S	86.5567	73.4471	52.6564	38.7157	86.5567	73.4471	52.6564	38.7157	26.8717
PianoL	CensusSGM_f_128_8P_NoS	86.5567	73.4471	52.6564	38.7157	86.5567	73.4471	52.6564	38.7157	26.8717
PianoL	CensusSGM_f_128_8P_S	86.5567	73.4471	52.6564	38.7157	86.5567	73.4471	52.6564	38.7157	26.8717
PianoL	**CensusSGM_f_64_4P_NoS**	**86.4358**	**73.1975**	**52.2387**	**38.2388**	**86.4358**	**73.1975**	**52.2387**	**38.2388**	27.3794
PianoL	CensusSGM_f_64_4P_S	86.4358	73.1975	52.2387	38.2388	86.4358	73.1975	52.2387	38.2388	27.3794
PianoL	CensusSGM_f_64_8P_NoS	86.4358	73.1975	52.2387	38.2388	86.4358	73.1975	52.2387	38.2388	27.3794

PianoL	CensusSGM_f_64_8P_S	86.4358	73.1975	52.2387	38.2388	86.4358	73.1975	52.2387	38.2388	27.3794
PianoL	CensusSGM_r_128_4P_NoS	94.3548	88.601	77.2225	59.0564	94.3548	88.601	77.2225	59.0564	28.7596
PianoL	CensusSGM_r_128_4P_S	94.3548	88.601	77.2225	59.0564	94.3548	88.601	77.2225	59.0564	28.7596
PianoL	CensusSGM_r_128_8P_NoS	94.3548	88.601	77.2225	59.0564	94.3548	88.601	77.2225	59.0564	28.7596
PianoL	CensusSGM_r_128_8P_S	94.3548	88.601	77.2225	59.0564	94.3548	88.601	77.2225	59.0564	28.7596
PianoL	CensusSGM_r_64_4P_NoS_C0	94.364	88.6211	77.2467	59.1385	94.364	88.6211	77.2467	59.1385	28.5255
PianoL	CensusSGM_r_64_4P_NoS	94.364	88.6211	77.2467	59.1385	94.364	88.6211	77.2467	59.1385	28.5255
PianoL	CensusSGM_r_64_4P_S	94.364	88.6211	77.2467	59.1385	94.364	88.6211	77.2467	59.1385	28.5255
PianoL	CensusSGM_r_64_8P_NoS	94.364	88.6211	77.2467	59.1385	94.364	88.6211	77.2467	59.1385	28.5255
PianoL	CensusSGM_r_64_8P_S	94.364	88.6211	77.2467	59.1385	94.364	88.6211	77.2467	59.1385	28.5255
Pipes	**CensusSGM_f_128_4P_NoS**	**82.1582**	**64.5574**	**35.714**	**23.0581**	**82.1582**	**64.5574**	**35.714**	**23.0581**	19.8703
Pipes	CensusSGM_f_128_4P_S	82.1582	64.5574	35.714	23.0581	82.1582	64.5574	35.714	23.0581	19.8703
Pipes	CensusSGM_f_128_8P_NoS	82.1582	64.5574	35.714	23.0581	82.1582	64.5574	35.714	23.0581	19.8703
Pipes	CensusSGM_f_128_8P_S	82.1582	64.5574	35.714	23.0581	82.1582	64.5574	35.714	23.0581	19.8703
Pipes	CensusSGM_f_64_4P_NoS	82.3856	65.016	36.5257	23.9993	82.3856	65.016	36.5257	23.9993	**19.806**
Pipes	CensusSGM_f_64_4P_S	82.3856	65.016	36.5257	23.9993	82.3856	65.016	36.5257	23.9993	19.806
Pipes	CensusSGM_f_64_8P_NoS	82.3856	65.016	36.5257	23.9993	82.3856	65.016	36.5257	23.9993	19.806
Pipes	CensusSGM_f_64_8P_S	82.3856	65.016	36.5257	23.9993	82.3856	65.016	36.5257	23.9993	19.806
Pipes	CensusSGM_r_128_4P_NoS	92.0137	84.2723	70.4508	46.6481	92.0137	84.2723	70.4508	46.6481	27.4115
Pipes	CensusSGM_r_128_4P_S	92.0137	84.2723	70.4508	46.6481	92.0137	84.2723	70.4508	46.6481	27.4115
Pipes	CensusSGM_r_128_8P_NoS	92.0137	84.2723	70.4508	46.6481	92.0137	84.2723	70.4508	46.6481	27.4115
Pipes	CensusSGM_r_128_8P_S	92.0137	84.2723	70.4508	46.6481	92.0137	84.2723	70.4508	46.6481	27.4115
Pipes	CensusSGM_r_64_4P_NoS_C0	92.0041	84.2578	70.4309	46.6253	92.0041	84.2578	70.4309	46.6253	27.3098
Pipes	CensusSGM_r_64_4P_NoS	92.0041	84.2578	70.4309	46.6253	92.0041	84.2578	70.4309	46.6253	27.3098
Pipes	CensusSGM_r_64_4P_S	92.0041	84.2578	70.4309	46.6253	92.0041	84.2578	70.4309	46.6253	27.3098
Pipes	CensusSGM_r_64_8P_NoS	92.0041	84.2578	70.4309	46.6253	92.0041	84.2578	70.4309	46.6253	27.3098
Pipes	CensusSGM_r_64_8P_S	92.0041	84.2578	70.4309	46.6253	92.0041	84.2578	70.4309	46.6253	27.3098
Playroom	**CensusSGM_f_128_4P_NoS**	**84.99**	**70.6916**	**47.3113**	**28.815**	**84.99**	**70.6916**	**47.3113**	**28.815**	**21.9782**
Playroom	CensusSGM_f_128_4P_S	84.99	70.6916	47.3113	28.815	84.99	70.6916	47.3113	28.815	21.9782
Playroom	CensusSGM_f_128_8P_NoS	84.99	70.6916	47.3113	28.815	84.99	70.6916	47.3113	28.815	21.9782
Playroom	CensusSGM_f_128_8P_S	84.99	70.6916	47.3113	28.815	84.99	70.6916	47.3113	28.815	21.9782
Playroom	CensusSGM_f_64_4P_NoS	85.7884	72.2121	49.9096	32.5253	85.7884	72.2121	49.9096	32.5253	23.6734
Playroom	CensusSGM_f_64_4P_S	85.7884	72.2121	49.9096	32.5253	85.7884	72.2121	49.9096	32.5253	23.6734
Playroom	CensusSGM_f_64_8P_NoS	85.7884	72.2121	49.9096	32.5253	85.7884	72.2121	49.9096	32.5253	23.6734
Playroom	CensusSGM_f_64_8P_S	85.7884	72.2121	49.9096	32.5253	85.7884	72.2121	49.9096	32.5253	23.6734
Playroom	CensusSGM_r_128_4P_NoS	93.2717	86.4942	73.1481	51.2254	93.2717	86.4942	73.1481	51.2254	24.5429
Playroom	CensusSGM_r_128_4P_S	93.2717	86.4942	73.1481	51.2254	93.2717	86.4942	73.1481	51.2254	24.5429
Playroom	CensusSGM_r_128_8P_NoS	93.2717	86.4942	73.1481	51.2254	93.2717	86.4942	73.1481	51.2254	24.5429
Playroom	CensusSGM_r_128_8P_S	93.2717	86.4942	73.1481	51.2254	93.2717	86.4942	73.1481	51.2254	24.5429
Playroom	CensusSGM_r_64_4P_NoS_C0	93.2715	86.4985	73.1516	51.2103	93.2715	86.4985	73.1516	51.2103	24.5012
Playroom	CensusSGM_r_64_4P_NoS	93.2715	86.4985	73.1516	51.2103	93.2715	86.4985	73.1516	51.2103	24.5012
Playroom	CensusSGM_r_64_4P_S	93.2715	86.4985	73.1516	51.2103	93.2715	86.4985	73.1516	51.2103	24.5012
Playroom	CensusSGM_r_64_8P_NoS	93.2715	86.4985	73.1516	51.2103	93.2715	86.4985	73.1516	51.2103	24.5012
Playroom	CensusSGM_r_64_8P_S	93.2715	86.4985	73.1516	51.2103	93.2715	86.4985	73.1516	51.2103	24.5012
Playtable	**CensusSGM_f_128_4P_NoS**	**84.2549**	**69.1299**	**45.3303**	**26.7732**	**84.2549**	**69.1299**	**45.3303**	**26.7732**	**18.521**
Playtable	CensusSGM_f_128_4P_S	84.2549	69.1299	45.3303	26.7732	84.2549	69.1299	45.3303	26.7732	18.521
Playtable	CensusSGM_f_128_8P_NoS	84.2549	69.1299	45.3303	26.7732	84.2549	69.1299	45.3303	26.7732	18.521
Playtable	CensusSGM_f_128_8P_S	84.2549	69.1299	45.3303	26.7732	84.2549	69.1299	45.3303	26.7732	18.521
Playtable	CensusSGM_f_64_4P_NoS	84.2986	69.208	45.4624	26.9518	84.2986	69.208	45.4624	26.9518	18.6695
Playtable	CensusSGM_f_64_4P_S	84.2986	69.208	45.4624	26.9518	84.2986	69.208	45.4624	26.9518	18.6695
Playtable	CensusSGM_f_64_8P_NoS	84.2986	69.208	45.4624	26.9518	84.2986	69.208	45.4624	26.9518	18.6695
Playtable	CensusSGM_f_64_8P_S	84.2986	69.208	45.4624	26.9518	84.2986	69.208	45.4624	26.9518	18.6695
Playtable	CensusSGM_r_128_4P_NoS	92.6806	85.3561	71.3637	49.1473	92.6806	85.3561	71.3637	49.1473	21.584
Playtable	CensusSGM_r_128_4P_S	92.6806	85.3561	71.3637	49.1473	92.6806	85.3561	71.3637	49.1473	21.584
Playtable	CensusSGM_r_128_8P_NoS	92.6806	85.3561	71.3637	49.1473	92.6806	85.3561	71.3637	49.1473	21.584

Playtable	CensusSGM_r_128_8P_S	92.6806	85.3561	71.3637	49.1473	92.6806	85.3561	71.3637	49.1473	21.584
Playtable	CensusSGM_r_64_4P_NoS_C0	92.6802	85.3543	71.3607	49.1427	92.6802	85.3543	71.3607	49.1427	21.5731
Playtable	CensusSGM_r_64_4P_NoS	92.6802	85.3543	71.3607	49.1427	92.6802	85.3543	71.3607	49.1427	21.5731
Playtable	CensusSGM_r_64_4P_S	92.6802	85.3543	71.3607	49.1427	92.6802	85.3543	71.3607	49.1427	21.5731
Playtable	CensusSGM_r_64_8P_NoS	92.6802	85.3543	71.3607	49.1427	92.6802	85.3543	71.3607	49.1427	21.5731
Playtable	CensusSGM_r_64_8P_S	92.6802	85.3543	71.3607	49.1427	92.6802	85.3543	71.3607	49.1427	21.5731
PlaytableP	**CensusSGM_f_128_4P_NoS**	**82.5094**	**65.4865**	**37.2077**	**21.9679**	**82.5094**	**65.4865**	**37.2077**	**21.9679**	**14.991**
PlaytableP	CensusSGM_f_128_4P_S	82.5094	65.4865	37.2077	21.9679	82.5094	65.4865	37.2077	21.9679	14.991
PlaytableP	CensusSGM_f_128_8P_NoS	82.5094	65.4865	37.2077	21.9679	82.5094	65.4865	37.2077	21.9679	14.991
PlaytableP	CensusSGM_f_128_8P_S	82.5094	65.4865	37.2077	21.9679	82.5094	65.4865	37.2077	21.9679	14.991
PlaytableP	CensusSGM_f_64_4P_NoS	82.5433	65.5501	37.326	22.1386	82.5433	65.5501	37.326	22.1386	15.1565
PlaytableP	CensusSGM_f_64_4P_S	82.5433	65.5501	37.326	22.1386	82.5433	65.5501	37.326	22.1386	15.1565
PlaytableP	CensusSGM_f_64_8P_NoS	82.5433	65.5501	37.326	22.1386	82.5433	65.5501	37.326	22.1386	15.1565
PlaytableP	CensusSGM_f_64_8P_S	82.5433	65.5501	37.326	22.1386	82.5433	65.5501	37.326	22.1386	15.1565
PlaytableP	CensusSGM_r_128_4P_NoS	92.3576	84.8056	69.9856	47.7516	92.3576	84.8056	69.9856	47.7516	20.6117
PlaytableP	CensusSGM_r_128_4P_S	92.3576	84.8056	69.9856	47.7516	92.3576	84.8056	69.9856	47.7516	20.6117
PlaytableP	CensusSGM_r_128_8P_NoS	92.3576	84.8056	69.9856	47.7516	92.3576	84.8056	69.9856	47.7516	20.6117
PlaytableP	CensusSGM_r_128_8P_S	92.3576	84.8056	69.9856	47.7516	92.3576	84.8056	69.9856	47.7516	20.6117
PlaytableP	CensusSGM_r_64_4P_NoS$_C$0	92.367	84.8133	69.9924	47.7498	92.367	84.8133	69.9924	47.7498	20.5748
PlaytableP	CensusSGM_r_64_4P_NoS	92.367	84.8133	69.9924	47.7498	92.367	84.8133	69.9924	47.7498	20.5748
PlaytableP	CensusSGM_r_64_4P_S	92.367	84.8133	69.9924	47.7498	92.367	84.8133	69.9924	47.7498	20.5748
PlaytableP	CensusSGM_r_64_8P_NoS	92.367	84.8133	69.9924	47.7498	92.367	84.8133	69.9924	47.7498	20.5748
PlaytableP	CensusSGM_r_64_8P_S	92.367	84.8133	69.9924	47.7498	92.367	84.8133	69.9924	47.7498	20.5748
Recycle	**CensusSGM_f_128_4P_NoS**	**81.2696**	**62.5462**	**32.3889**	**16.4465**	**81.2696**	**62.5462**	**32.3889**	**16.4465**	**9.38382**
Recycle	CensusSGM_f_128_4P_S	81.2696	62.5462	32.3889	16.4465	81.2696	62.5462	32.3889	16.4465	9.38382
Recycle	CensusSGM_f_128_8P_NoS	81.2696	62.5462	32.3889	16.4465	81.2696	62.5462	32.3889	16.4465	9.38382
Recycle	CensusSGM_f_128_8P_S	81.2696	62.5462	32.3889	16.4465	81.2696	62.5462	32.3889	16.4465	9.38382
Recycle	CensusSGM_f_64_4P_NoS	81.3042	62.6232	32.53	16.6206	81.3042	62.6232	32.53	16.6206	9.40765
Recycle	CensusSGM_f_64_4P_S	81.3042	62.6232	32.53	16.6206	81.3042	62.6232	32.53	16.6206	9.40765
Recycle	CensusSGM_f_64_8P_NoS	81.3042	62.6232	32.53	16.6206	81.3042	62.6232	32.53	16.6206	9.40765
Recycle	CensusSGM_f_64_8P_S	81.3042	62.6232	32.53	16.6206	81.3042	62.6232	32.53	16.6206	9.40765
Recycle	CensusSGM_r_128_4P_NoS	92.0831	83.8639	67.7573	39.0506	92.0831	83.8639	67.7573	39.0506	10.9176
Recycle	CensusSGM_r_128_4P_S	92.0831	83.8639	67.7573	39.0506	92.0831	83.8639	67.7573	39.0506	10.9176
Recycle	CensusSGM_r_128_8P_NoS	92.0831	83.8639	67.7573	39.0506	92.0831	83.8639	67.7573	39.0506	10.9176
Recycle	CensusSGM_r_128_8P_S	92.0831	83.8639	67.7573	39.0506	92.0831	83.8639	67.7573	39.0506	10.9176
Recycle	CensusSGM_r_64_4P_NoS_C0	92.0839	83.8659	67.7598	39.0567	92.0839	83.8659	67.7598	39.0567	10.9219
Recycle	CensusSGM_r_64_4P_NoS	92.0839	83.8659	67.7598	39.0567	92.0839	83.8659	67.7598	39.0567	10.9219
Recycle	CensusSGM_r_64_4P_S	92.0839	83.8659	67.7598	39.0567	92.0839	83.8659	67.7598	39.0567	10.9219
Recycle	CensusSGM_r_64_8P_NoS	92.0839	83.8659	67.7598	39.0567	92.0839	83.8659	67.7598	39.0567	10.9219
Recycle	CensusSGM_r_64_8P_S	92.0839	83.8659	67.7598	39.0567	92.0839	83.8659	67.7598	39.0567	10.9219
Shelves	**CensusSGM_f_128_4P_NoS**	**88.4344**	**77.4158**	**59.6269**	**46.2371**	**88.4344**	**77.4158**	**59.6269**	**46.2371**	23.2456
Shelves	CensusSGM_f_128_4P_S	88.4344	77.4158	59.6269	46.2371	88.4344	77.4158	59.6269	46.2371	23.2456
Shelves	CensusSGM_f_128_8P_NoS	88.4344	77.4158	59.6269	46.2371	88.4344	77.4158	59.6269	46.2371	23.2456
Shelves	CensusSGM_f_128_8P_S	88.4344	77.4158	59.6269	46.2371	88.4344	77.4158	59.6269	46.2371	23.2456
Shelves	CensusSGM_f_64_4P_NoS	88.4496	77.453	59.6684	46.2803	88.4496	77.453	59.6684	46.2803	**23.1585**
Shelves	CensusSGM_f_64_4P_S	88.4496	77.453	59.6684	46.2803	88.4496	77.453	59.6684	46.2803	23.1585
Shelves	CensusSGM_f_64_8P_NoS	88.4496	77.453	59.6684	46.2803	88.4496	77.453	59.6684	46.2803	23.1585
Shelves	CensusSGM_f_64_8P_S	88.4496	77.453	59.6684	46.2803	88.4496	77.453	59.6684	46.2803	23.1585
Shelves	CensusSGM_r_128_4P_NoS	94.8281	89.751	80.0644	60.6749	94.8281	89.751	80.0644	60.6749	16.8283
Shelves	CensusSGM_r_128_4P_S	94.8281	89.751	80.0644	60.6749	94.8281	89.751	80.0644	60.6749	16.8283
Shelves	CensusSGM_r_128_8P_NoS	94.8281	89.751	80.0644	60.6749	94.8281	89.751	80.0644	60.6749	16.8283
Shelves	CensusSGM_r_128_8P_S	94.8281	89.751	80.0644	60.6749	94.8281	89.751	80.0644	60.6749	16.8283
Shelves	CensusSGM_r_64_4P_NoS_C0	94.829	89.7514	80.0662	60.6791	94.829	89.7514	80.0662	60.6791	16.8095
Shelves	CensusSGM_r_64_4P_NoS	94.829	89.7514	80.0662	60.6791	94.829	89.7514	80.0662	60.6791	16.8095
Shelves	CensusSGM_r_64_4P_S	94.829	89.7514	80.0662	60.6791	94.829	89.7514	80.0662	60.6791	16.8095

Shelves	CensusSGM_r_64_8P_NoS	94.829	89.7514	80.0662	60.6791	94.829	89.7514	80.0662	60.6791	16.8095
Shelves	CensusSGM_r_64_8P_S	94.829	89.7514	80.0662	60.6791	94.829	89.7514	80.0662	60.6791	16.8095
Teddy	CensusSGM_f_128_4P_NoS	82.4743	42.2956	19.4029	13.1962	82.4743	42.2956	19.4029	13.1962	9.06893
Teddy	CensusSGM_f_128_4P_S	82.4743	42.2956	19.4029	13.1962	82.4743	42.2956	19.4029	13.1962	9.06893
Teddy	CensusSGM_f_128_8P_NoS	82.4743	42.2956	19.4029	13.1962	82.4743	42.2956	19.4029	13.1962	9.06893
Teddy	CensusSGM_f_128_8P_S	82.4743	42.2956	19.4029	13.1962	82.4743	42.2956	19.4029	13.1962	9.06893
Teddy	**CensusSGM_f_64_4P_NoS**	**82.4729**	**42.2932**	**19.398**	**13.1907**	**82.4729**	**42.2932**	**19.398**	**13.1907**	**9.06159**
Teddy	CensusSGM_f_64_4P_S	82.4729	42.2932	19.398	13.1907	82.4729	42.2932	19.398	13.1907	9.06159
Teddy	CensusSGM_f_64_8P_NoS	82.4729	42.2932	19.398	13.1907	82.4729	42.2932	19.398	13.1907	9.06159
Teddy	CensusSGM_f_64_8P_S	82.4729	42.2932	19.398	13.1907	82.4729	42.2932	19.398	13.1907	9.06159
Teddy	CensusSGM_r$_2$8_4P_NoS	87.5646	59.1575	36.8726	20.8878	87.5646	59.1575	36.8726	20.8878	10.5069
Teddy	CensusSGM_r_128_4P_S	87.5646	59.1575	36.8726	20.8878	87.5646	59.1575	36.8726	20.8878	10.5069
Teddy	CensusSGM_r_128_8P_NoS	87.5646	59.1575	36.8726	20.8878	87.5646	59.1575	36.8726	20.8878	10.5069
Teddy	CensusSGM_r_128_8P_S	87.5646	59.1575	36.8726	20.8878	87.5646	59.1575	36.8726	20.8878	10.5069
Teddy	CensusSGM_r_64_4P_NoS_C0	87.5642	59.1562	36.8711	20.882	87.5642	59.1562	36.8711	20.882	10.4977
Teddy	CensusSGM_r_64_4P_NoS	87.5642	59.1562	36.8711	20.882	87.5642	59.1562	36.8711	20.882	10.4977
Teddy	CensusSGM_r_64_4P_S	87.5642	59.1562	36.8711	20.882	87.5642	59.1562	36.8711	20.882	10.4977
Teddy	CensusSGM_r_64_8P_NoS	87.5642	59.1562	36.8711	20.882	87.5642	59.1562	36.8711	20.882	10.4977
Teddy	CensusSGM_r_64_8P_S	87.5642	59.1562	36.8711	20.882	87.5642	59.1562	36.8711	20.882	10.4977
Vintage	**CensusSGM_f_128_4P_NoS**	**86.8344**	**73.9693**	**52.449**	**36.0391**	**86.8344**	**73.9693**	**52.449**	**36.0391**	**39.4277**
Vintage	CensusSGM_f_128_4P_S	86.8344	73.9693	52.449	36.0391	86.8344	73.9693	52.449	36.0391	39.4277
Vintage	CensusSGM_f_128_8P_NoS	86.8344	73.9693	52.449	36.0391	86.8344	73.9693	52.449	36.0391	39.4277
Vintage	CensusSGM_f_128_8P_S	86.8344	73.9693	52.449	36.0391	86.8344	73.9693	52.449	36.0391	39.4277
Vintage	CensusSGM_f_64_4P_NoS	93.1837	86.4411	75.2787	67.7643	93.1837	86.4411	75.2787	67.7643	106.288
Vintage	CensusSGM_f_64_4P_S	93.1837	86.4411	75.2787	67.7643	93.1837	86.4411	75.2787	67.7643	106.288
Vintage	CensusSGM_f_64_8P_NoS	93.1837	86.4411	75.2787	67.7643	93.1837	86.4411	75.2787	67.7643	106.288
Vintage	CensusSGM_f_64_8P_S	93.1837	86.4411	75.2787	67.7643	93.1837	86.4411	75.2787	67.7643	106.288
Vintage	CensusSGM_r_128_4P_NoS	93.9155	87.9549	76.0136	55.3263	93.9155	87.9549	76.0136	55.3263	37.3618
Vintage	CensusSGM_r_128_4P_S	93.9155	87.9549	76.0136	55.3263	93.9155	87.9549	76.0136	55.3263	37.3618
Vintage	CensusSGM_r_128_8P_NoS	93.9155	87.9549	76.0136	55.3263	93.9155	87.9549	76.0136	55.3263	37.3618
Vintage	CensusSGM_r_128_8P_S	93.9155	87.9549	76.0136	55.3263	93.9155	87.9549	76.0136	55.3263	37.3618
Vintage	CensusSGM_r_64_4P_NoS_C0	93.9136	87.9511	76.0132	55.3247	93.9136	87.9511	76.0132	55.3247	37.3693
Vintage	CensusSGM_r_64_4P_NoS	93.9136	87.9511	76.0132	55.3247	93.9136	87.9511	76.0132	55.3247	37.3693
Vintage	CensusSGM_r_64_4P_S	93.9136	87.9511	76.0132	55.3247	93.9136	87.9511	76.0132	55.3247	37.3693
Vintage	CensusSGM_r_64_8P_NoS	93.9136	87.9511	76.0132	55.3247	93.9136	87.9511	76.0132	55.3247	37.3693
Vintage	CensusSGM_r_64_8P_S	93.9136	87.9511	76.0132	55.3247	93.9136	87.9511	76.0132	55.3247	37.3693

TABLE V: CelsusSGM Ablation study with Middlebury Benchmark

dataset	algorithm	Time	FPS
Adirondack	CensusSGM_f_128_4P_NoS	517.773 [ms]	1.93135 [fps]
Adirondack	CensusSGM_f_128_4P_S	613.103 [ms]	1.63105 [fps]
Adirondack	CensusSGM_f_128_8P_NoS	630.187 [ms]	1.58683 [fps]
Adirondack	CensusSGM_f_128_8P_S	405.67 [ms]	2.46506 [fps]
Adirondack	CensusSGM_f_64_4P_NoS	297.129 [ms]	3.36554 [fps]
Adirondack	CensusSGM_f_64_4P_S	524.599 [ms]	1.90622 [fps]
Adirondack	CensusSGM_f_64_8P_NoS	254.878 [ms]	3.92345 [fps]
Adirondack	CensusSGM_f_64_8P_S	263.069 [ms]	3.80128 [fps]
Adirondack	CensusSGM_r_128_4P_NoS	202.562 [ms]	4.93676 [fps]
Adirondack	CensusSGM_r_128_4P_S	179.552 [ms]	5.56942 [fps]
Adirondack	CensusSGM_r_128_8P_NoS	189.456 [ms]	5.27827 [fps]
Adirondack	CensusSGM_r_128_8P_S	189.985 [ms]	5.26357 [fps]
Adirondack	CensusSGM_r_64_4P_NoS_C0	102.033 [ms]	9.80075 [fps]
Adirondack	**CensusSGM_r_64_4P_NoS**	101.787 [ms]	**9.82444 [fps]**
Adirondack	**CensusSGM_r_64_4P_S**	**101.488 [ms]**	9.85338 [fps]

Adirondack	CensusSGM_r_64_8P_NoS	101.622 [ms]	9.84039 [fps]
Adirondack	CensusSGM_r_64_8P_S	101.759 [ms]	82.0724
ArtL	CensusSGM_f_128_4P_NoS	367.633 [ms]	2.7201 [fps]
ArtL	CensusSGM_f_128_4P_S	369.772 [ms]	2.70437 [fps]
ArtL	CensusSGM_f_128_8P_NoS	357.11 [ms]	2.80026 [fps]
ArtL	CensusSGM_f_128_8P_S	356.488 [ms]	2.80514 [fps]
ArtL	CensusSGM_f_64_4P_NoS	165.704 [ms]	6.03486 [fps]
ArtL	CensusSGM_f_64_4P_S	343.638 [ms]	2.91004 [fps]
ArtL	CensusSGM_f_64_8P_NoS	183.525 [ms]	5.44885 [fps]
ArtL	CensusSGM_f_64_8P_S	186.958 [ms]	5.34879 [fps]
ArtL	CensusSGM_r_128_4P_NoS	113.53 [ms]	8.80824 [fps]
ArtL	CensusSGM_r_128_4P_S	113.199 [ms]	8.834 [fps]
ArtL	CensusSGM_r_128_8P_NoS	189.905 [ms]	5.26579 [fps]
ArtL	CensusSGM_r_128_8P_S	189.889 [ms]	5.26623 [fps]
ArtL	CensusSGM_r_64_4P_NoS_C0	101.811 [ms]	9.82212 [fps]
ArtL	**CensusSGM_r_64_4P_NoS**	**60.357 [ms]**	**16.5681 [fps]**
ArtL	CensusSGM_r_64_4P_S	77.736 [ms]	12.8641 [fps]
ArtL	CensusSGM_r_64_8P_NoS	101.437 [ms]	9.85834 [fps]
ArtL	CensusSGM_r_64_8P_S	101.405 [ms]	9.86145 [fps]
Jadeplant	CensusSGM_f_128_4P_NoS	397.887 [ms]	2.51328 [fps]
Jadeplant	CensusSGM_f_128_4P_S	397.164 [ms]	2.51785 [fps]
Jadeplant	CensusSGM_f_128_8P_NoS	362.118 [ms]	2.76153 [fps]
Jadeplant	CensusSGM_f_128_8P_S	357.34 [ms]	2.79846 [fps]
Jadeplant	CensusSGM_f_64_4P_NoS	162.375 [ms]	6.15858 [fps]
Jadeplant	CensusSGM_f_64_4P_S	248.823 [ms]	4.01892 [fps]
Jadeplant	CensusSGM_f_64_8P_NoS	186.149 [ms]	5.37204 [fps]
Jadeplant	CensusSGM_f_64_8P_S	185.186 [ms]	5.39998 [fps]
Jadeplant	CensusSGM_r_128_4P_NoS	113.399 [ms]	8.81842 [fps]
Jadeplant	CensusSGM_r_128_4P_S	104.182 [ms]	9.59859 [fps]
Jadeplant	CensusSGM_r_128_8P_NoS	113.255 [ms]	8.82963 [fps]
Jadeplant	CensusSGM_r_128_8P_S	113.263 [ms]	8.82901 [fps]
Jadeplant	CensusSGM_r_64_4P_NoS_C0	61.273 [ms]	16.3204 [fps]
Jadeplant	CensusSGM_r_64_4P_NoS	60.6 [ms]	16.5017 [fps]
Jadeplant	**CensusSGM_r_64_4P_S**	**60.497 [ms]**	**16.5297 [fps]**
Jadeplant	CensusSGM_r_64_8P_NoS	62.64 [ms]	15.9642 [fps]
Jadeplant	CensusSGM_r_64_8P_S	101.678 [ms]	9.83497 [fps]
Motorcycle	CensusSGM_f_128_4P_NoS	355.956 [ms]	2.80934 [fps]
Motorcycle	CensusSGM_f_128_4P_S	348.619 [ms]	2.86846 [fps]
Motorcycle	CensusSGM_f_128_8P_NoS	348.367 [ms]	2.87054 [fps]
Motorcycle	CensusSGM_f_128_8P_S	345.169 [ms]	2.89713 [fps]
Motorcycle	CensusSGM_f_64_4P_NoS	157.024 [ms]	6.36845 [fps]
Motorcycle	CensusSGM_f_64_4P_S	175.085 [ms]	5.71151 [fps]
Motorcycle	CensusSGM_f_64_8P_NoS	182.41 [ms]	5.48216 [fps]
Motorcycle	CensusSGM_f_64_8P_S	165.719 [ms]	6.03431 [fps]
Motorcycle	CensusSGM_r_128_4P_NoS	113.424 [ms]	8.81648 [fps]
Motorcycle	CensusSGM_r_128_4P_S	78.907 [ms]	12.6731 [fps]
Motorcycle	CensusSGM_r_128_8P_NoS	88.484 [ms]	11.3015 [fps]
Motorcycle	CensusSGM_r_128_8P_S	114.015 [ms]	8.77078 [fps]
Motorcycle	CensusSGM_r_64_4P_NoS_C0	60.814 [ms]	16.4436 [fps]
Motorcycle	**CensusSGM_r_64_4P_NoS**	**46.602 [ms]**	**21.4583 [fps]**
Motorcycle	CensusSGM_r_64_4P_S	60.212 [ms]	16.608 [fps]
Motorcycle	CensusSGM_r_64_8P_NoS	61.456 [ms]	16.2718 [fps]
Motorcycle	CensusSGM_r_64_8P_S	60.359 [ms]	16.5675 [fps]
MotorcycleE	CensusSGM_f_128_P_NoS	352.042 [ms]	2.84057 [fps]
MotorcycleE	CensusSGM_f_128_4P_S	351.548 [ms]	2.84456 [fps]

MotorcycleE	CensusSGM_f_128_8P_NoS	346.657 [ms]	2.8847 [fps]
MotorcycleE	CensusSGM_f_128_8P_S	344.704 [ms]	2.90104 [fps]
MotorcycleE	CensusSGM_f_64_4P_NoS	154.312 [ms]	6.48038 [fps]
MotorcycleE	CensusSGM_f_64_4P_S	172.881 [ms]	5.78433 [fps]
MotorcycleE	CensusSGM_f_64_8P_NoS	163.77 [ms]	6.10612 [fps]
MotorcycleE	CensusSGM_f_64_8P_S	159.069 [ms]	6.28658 [fps]
MotorcycleE	CensusSGM_r_128_4P_NoS	123.317 [ms]	8.10918 [fps]
MotorcycleE	CensusSGM_r_128_4P_S	75.554 [ms]	13.2356 [fps]
MotorcycleE	CensusSGM_r_128_8P_NoS	88.566 [ms]	11.291 [fps]
MotorcycleE	CensusSGM_r_128_8P_S	88.065 [ms]	11.3552 [fps]
MotorcycleE	CensusSGM_r_64_4P_NoS_C0	60.843 [ms]	16.4357 [fps]
MotorcycleE	**CensusSGM_r_64_4P_NoS**	**46.521 [ms]**	**21.4957 [fps]**
MotorcycleE	CensusSGM_r_64_4P_S	69.392 [ms]	14.4109 [fps]
MotorcycleE	CensusSGM_r_64_8P_NoS	60.313 [ms]	16.5802 [fps]
MotorcycleE	CensusSGM_r_64_8P_S	60.686 [ms]	16.4783 [fps]
Piano	CensusSGM_f_128_4P_NoS	320.049 [ms]	3.12452 [fps]
Piano	CensusSGM_f_128_4P_S	309.497 [ms]	3.23105 [fps]
Piano	CensusSGM_f_128_8P_NoS	346.657 [ms]	2.8847 [fps]
Piano	CensusSGM_f_128_8P_S	308.571 [ms]	3.24075 [fps]
Piano	CensusSGM_f_64_4P_NoS	143.996 [ms]	6.94464 [fps]
Piano	CensusSGM_f_64_4P_S	146.815 [ms]	6.81129 [fps]
Piano	CensusSGM_f_64_8P_NoS	156.501 [ms]	6.38974 [fps]
Piano	CensusSGM_f_64_8P_S	146.118 [ms]	6.84378 [fps]
Piano	CensusSGM_r_128_4P_NoS	88.128 [ms]	11.3471 [fps]
Piano	CensusSGM_r_128_4P_S	75.66 [ms]	13.217 [fps]
Piano	CensusSGM_r_128_8P_NoS	88.649 [ms]	11.2804 [fps]
Piano	CensusSGM_r_128_8P_S	75.504 [ms]	13.2443 [fps]
Piano	CensusSGM_r_64_4P_NoS_C0	60.425 [ms]	16.5494 [fps]
Piano	**CensusSGM_r_64_4P_NoS**	**46.644 [ms]**	**21.439 [fps]**
Piano	CensusSGM_r_64_4P_S	46.79 [ms]	21.3721 [fps]
Piano	CensusSGM_r_64_8P_NoS	60.424 [ms]	16.5497 [fps]
Piano	CensusSGM_r_64_8P_S	46.701 [ms]	21.4128 [fps]
PianoL	CensusSGM_f_128_4P_NoS	344.953 [ms]	2.89895 [fps]
PianoL	CensusSGM_f_128_4P_S	336.241 [ms]	2.97406 [fps]
PianoL	CensusSGM_f_128_8P_NoS	340.171 [ms]	2.9397 [fps]
PianoL	CensusSGM_f_128_8P_S	340.466 [ms]	2.93715 [fps]
PianoL	CensusSGM_f_64_4P_NoS	158.755 [ms]	6.29901 [fps]
PianoL	CensusSGM_f_64_4P_S	166.332 [ms]	6.01207 [fps]
PianoL	CensusSGM_f_64_8P_NoS	161.09 [ms]	6.20771 [fps]
PianoL	CensusSGM_f_64_8P_S	155.113 [ms]	6.44691 [fps]
PianoL	CensusSGM_r_128_4P_NoS	88.944 [ms]	11.243 [fps]
PianoL	CensusSGM_r_128_4P_S	68.002 [ms]	14.7054 [fps]
PianoL	CensusSGM_r_128_8P_NoS	88.261 [ms]	11.33 [fps]
PianoL	CensusSGM_r_128_8P_S	75.521 [ms]	13.2414 [fps]
PianoL	CensusSGM_r_64_4P_NoS_C0	60.373 [ms]	16.5637 [fps]
PianoL	**CensusSGM_r_64_4P_NoS**	**46.433 [ms]**	**21.5364 [fps]**
PianoL	CensusSGM_r_64_4P_S	46.927 [ms]	21.3097 [fps]
PianoL	CensusSGM_r_64_8P_NoS	46.468 [ms]	21.5202 [fps]
PianoL	CensusSGM_r_64_8P_S	46.508 [ms]	21.5017 [fps]
Pipes	CensusSGM_f_128_4P_NoS	144.35 [ms]	6.92761[fps]
Pipes	CensusSGM_f_128_4P_S	143.074 [ms]	6.98939 [fps]
Pipes	CensusSGM_f_128_8P_NoS	143.487 [ms]	6.96927 [fps]
Pipes	CensusSGM_f_128_8P_S	143.284 [ms]	6.97915 [fps]
Pipes	CensusSGM_f_64_4P_NoS	74.113 [ms]	13.4929 [fps]
Pipes	CensusSGM_f_64_4P_S	74.183 [ms]	13.4802 [fps]

Pipes	CensusSGM_f_64_8P_NoS	84.554 [ms]	11.8268 [fps]
Pipes	CensusSGM_f_64_8P_S	74.278 [ms]	13.4629 [fps]
Pipes	CensusSGM_r_128_4P_NoS	88.893 [ms]	11.2495 [fps]
Pipes	CensusSGM_r_128_4P_S	67.93 [ms]	14.721 [fps]
Pipes	CensusSGM_r_128_8P_NoS	95.223 [ms]	10.5017 [fps]
Pipes	CensusSGM_r_128_8P_S	68.093 [ms]	14.6858 [fps]
Pipes	CensusSGM_r_64_4P_NoS_C0	61.425 [ms]	16.28 [fps]
Pipes	CensusSGM_r_64_4P_NoS	47.007 [ms]	21.2734 [fps]
Pipes	CensusSGM_r_64_4P_S	46.85 [ms]	21.3447 [fps]
Pipes	**CensusSGM_r_64_8P_NoS**	**39.972 [ms]**	**25.0175 [fps]**
Pipes	CensusSGM_r_64_8P_S	47.131 [ms]	21.2175 [fps]
Playroom	CensusSGM_f_128_4P_NoS	340.375 [ms]	2.93794 [fps]
Playroom	CensusSGM_f_128_4P_S	329.391 [ms]	3.03591 [fps]
Playroom	CensusSGM_f_128_8P_NoS	334.623 [ms]	2.98844 [fps]
Playroom	CensusSGM_f_128_8P_S	332.525 [ms]	3.00729 [fps]
Playroom	CensusSGM_f_64_4P_NoS	150.899 [ms]	6.62695 [fps]
Playroom	CensusSGM_f_64_4P_S	150.251 [ms]	6.65553 [fps]
Playroom	CensusSGM_f_64_8P_NoS	150.236 [ms]	6.65619 [fps]
Playroom	CensusSGM_f_64_8P_S	151.596 [ms]	6.59648 [fps]
Playroom	CensusSGM_r_128_4P_NoS	88.428 [ms]	11.3086 [fps]
Playroom	CensusSGM_r_128_4P_S	68.026 [ms]	14.7003 [fps]
Playroom	CensusSGM_r_128_8P_NoS	75.561 [ms]	13.2343 [fps]
Playroom	CensusSGM_r_128_8P_S	68.781 [ms]	14.5389 [fps]
Playroom	CensusSGM_r_64_4P_NoS_C0	53.272 [ms]	18.7716 [fps]
Playroom	CensusSGM_r_64_4P_NoS	46.762 [ms]	21.3849 [fps]
Playroom	CensusSGM_r_64_4P_S	47.099 [ms]	21.2319 [fps]
Playroom	**CensusSGM_r_64_8P_NoS**	**39.797 [ms]**	**25.1275 [fps]**
Playroom	CensusSGM_r_64_8P_S	46.858 [ms]	21.3411 [fps]
Playtable	CensusSGM_f_128_4P_NoS	301.109 [ms]	3.32106 [fps]
Playtable	CensusSGM_f_128_4P_S	293.734 [ms]	3.40444 [fps]
Playtable	CensusSGM_f_128_8P_NoS	295.786 [ms]	3.38082 [fps]
Playtable	CensusSGM_f_128_8P_S	294.27 [ms]	3.39824 [fps]
Playtable	CensusSGM_f_64_4P_NoS	141.06 [ms]	7.08918 [fps]
Playtable	CensusSGM_f_64_4P_S	137.334 [ms]	7.28152 [fps]
Playtable	CensusSGM_f_64_8P_NoS	150.236 [ms]	6.65619 [fps]
Playtable	CensusSGM_f_64_8P_S	137.491 [ms]	7.2732 [fps]
Playtable	CensusSGM_r_128_4P_NoS	76.377 [ms]	13.0929 [fps]
Playtable	CensusSGM_r_128_4P_S	68.026 [ms]	14.7003 [fps]
Playtable	CensusSGM_r_128_8P_NoS	68.459 [ms]	14.6073 [fps]
Playtable	CensusSGM_r_128_8P_S	67.995 [ms]	14.707 [fps]
Playtable	CensusSGM_r_64_4P_NoS_C0	47.027 [ms]	21.2644 [fps]
Playtable	CensusSGM_r_64_4P_NoS	46.905 [ms]	21.3197 [fps]
Playtable	CensusSGM_r_64_4P_S	46.863 [ms]	21.3388 [fps]
Playtable	**CensusSGM_r_64_8P_NoS**	**39.81 [ms]**	**25.1193 [fps]**
Playtable	CensusSGM_r_64_8P_S	47.012 [ms]	21.2712 [fps]
PlaytableP	CensusSGM_f_128_4P_NoS	369.618 [ms]	2.7055 [fps]
PlaytableP	CensusSGM_f_128_4P_S	359.458 [ms]	2.78197 [fps]
PlaytableP	CensusSGM_f_128_8P_NoS	361.109 [ms]	2.76925 [fps]
PlaytableP	CensusSGM_f_128_8P_S	367.561 [ms]	2.72064 [fps]
PlaytableP	CensusSGM_f_64_4P_NoS	161.564 [ms]	6.1895 [fps]
PlaytableP	CensusSGM_f_64_4P_S	168.903 [ms]	5.92056 [fps]
PlaytableP	CensusSGM_f_64_8P_NoS	162.27 [ms]	6.16257 [fps]
PlaytableP	CensusSGM_f_64_8P_S	165.936 [ms]	6.02642 [fps]
PlaytableP	CensusSGM_r_128_4P_NoS	68.37 [ms]	14.6263 [fps]
PlaytableP	CensusSGM_r_128_4P_S	67.847 [ms]	14.739 [fps]

PlaytableP	CensusSGM_r_128_8P_NoS	68.059 [ms]	14.6931 [fps]
PlaytableP	CensusSGM_r_128_8P_S	67.817 [ms]	14.7456 [fps]
PlaytableP	CensusSGM_r_64_4P_NoS$_C$0	47.205 [ms]	21.1842 [fps]
PlaytableP	CensusSGM_r_64_4P_NoS	47.285 [ms]	21.1484 [fps]
PlaytableP	CensusSGM_r_64_4P_S	46.877 [ms]	21.3324 [fps]
PlaytableP	**CensusSGM_r_64_8P_NoS**	**39.996 [ms]**	**25.0025 [fps]**
PlaytableP	CensusSGM_r_64_8P_S	46.789 [ms]	21.3725 [fps]
Recycle	CensusSGM_f_128_4P_NoS	324.912 [ms]	3.07776 [fps]
Recycle	CensusSGM_f_128_4P_S	314.668 [ms]	3.17795 [fps]
Recycle	CensusSGM_f_128_8P_NoS	314.97 [ms]	3.17491 [fps]
Recycle	CensusSGM_f_128_8P_S	318.043 [ms]	3.14423 [fps]
Recycle	CensusSGM_f_64_4P_NoS	145.699 [ms]	6.86347 [fps]
Recycle	CensusSGM_f_64_4P_S	147.061 [ms]	6.7999 [fps]
Recycle	CensusSGM_f_64_8P_NoS	144.555 [ms]	6.91778 [fps]
Recycle	CensusSGM_f_64_8P_S	145.081 [ms]	6.8927 [fps]
Recycle	CensusSGM_r_128_4P_NoS	68.188 [ms]	14.6653 [fps]
Recycle	CensusSGM_r_128_4P_S	64.001 [ms]	15.6248 [fps]
Recycle	CensusSGM_r_128_8P_NoS	68.653 [ms]	14.566 [fps]
Recycle	CensusSGM_r_128_8P_S	67.8 [ms]	14.7493 [fps]
Recycle	CensusSGM_r_64_4P_NoS_C0	47.027 [ms]	21.2644 [fps]
Recycle	CensusSGM_r_64_4P_NoS	47.243 [ms]	21.1672 [fps]
Recycle	CensusSGM_r_64_4P_S	46.929 [ms]	21.3088 [fps]
Recycle	**CensusSGM_r_64_8P_NoS**	**40.211 [ms]**	**24.8688 [fps]**
Recycle	CensusSGM_r_64_8P_S	46.9 [ms]	21.322[fps]
Shelves	CensusSGM_f_128_4P_NoS	372.228 [ms]	2.68653 [fps]
Shelves	CensusSGM_f_128_4P_S	360.33 [ms]	2.77523 [fps]
Shelves	CensusSGM_f_128_8P_NoS	374.262 [ms]	2.67193 [fps]
Shelves	CensusSGM_f_128_8P_S	359.506 [ms]	2.78159 [fps]
Shelves	CensusSGM_f_64_4P_NoS	160.068 [ms]	6.24734 [fps]
Shelves	CensusSGM_f_64_4P_S	164.956 [ms]	6.06222 [fps]
Shelves	CensusSGM_f_64_8P_NoS	163.298 [ms]	6.12377 [fps]
Shelves	CensusSGM_f_64_8P_S	145.081 [ms]	6.8927 [fps]
Shelves	CensusSGM_r_128_4P_NoS	62.968 [ms]	15.8811 [fps]
Shelves	CensusSGM_r_128_4P_S	63.3 [ms]	15.7978 [fps]
Shelves	CensusSGM_r_128_8P_NoS	68.704 [ms]	14.5552 [fps]
Shelves	CensusSGM_r_128_8P_S	69.092 [ms]	14.4735 [fps]
Shelves	CensusSGM_r_64_4P_NoS_C0	53.561 [ms]	18.6703 [fps]
Shelves	CensusSGM_r_64_4P_NoS	46.92 [ms]	21.3129 [fps]
Shelves	CensusSGM_r_64_4P_S	46.884 [ms]	21.3292 [fps]
Shelves	**CensusSGM_r_64_8P_NoS**	**40.534 [ms]**	**24.6706 [fps]**
Shelves	CensusSGM_r_64_8P_S	47.065 [ms]	21.2472 [fps]
Teddy	CensusSGM_f_128_4P_NoS	81.749 [ms]	12.2326 [fps]
Teddy	CensusSGM_f_128_4P_S	82.336 [ms]	12.1454 [fps]
Teddy	CensusSGM_f_128_8P_NoS	168.674 [ms]	5.9286 [fps]
Teddy	CensusSGM_f_128_8P_S	82.642 [ms]	12.1004 [fps]
Teddy	CensusSGM_f_64_4P_NoS	42.875 [ms]	23.3236 [fps]
Teddy	CensusSGM_f_64_4P_S	43.097 [ms]	23.2035 [fps]
Teddy	CensusSGM_f_64_8P_NoS	43.189 [ms]	23.154 [fps]
Teddy	CensusSGM_f_64_8P_S	42.631 [ms]	23.4571 [fps]
Teddy	CensusSGM_r$_2$8_$4P_NoS$	63.243 [ms]	15.812 [fps]
Teddy	CensusSGM_r_128_4P_S	62.919 [ms]	15.8935 [fps]
Teddy	CensusSGM_r_128_8P_NoS	68.133 [ms]	14.6772 [fps]
Teddy	CensusSGM_r_128_8P_S	67.917 [ms]	14.7239 [fps]
Teddy	CensusSGM_r_64_4P_NoS_C0	46.907 [ms]	21.3188 [fps]
Teddy	CensusSGM_r_64_4P_NoS	46.989 [ms]	21.2816 [fps]

Teddy	CensusSGM_r_64_4P_S	47.069 [ms]	21.2454 [fps]
Teddy	**CensusSGM_r_64_8P_NoS**	**40.592 [ms]**	**24.6354 [fps]**
Teddy	CensusSGM_r_64_8P_S	47.268 [ms]	21.156 [fps]
Vintage	CensusSGM_f_128_4P_NoS	337.415 [ms]	2.96371 [fps]
Vintage	CensusSGM_f_128_4P_S	328.073 [ms]	3.0481 [fps]
Vintage	CensusSGM_f_128_8P_NoS	387.558 [ms]	2.58026 [fps]
Vintage	CensusSGM_f_128_8P_S	326.109 [ms]	3.06646 [fps]
Vintage	CensusSGM_f_64_4P_NoS	153.863 [ms]	6.49929 [fps]
Vintage	CensusSGM_f_64_4P_S	148.993 [ms]	6.71172 [fps]
Vintage	CensusSGM_f_64_8P_NoS	149.104 [ms]	6.70673 [fps]
Vintage	CensusSGM_f_64_8P_S	152.44 [ms]	6.55996 [fps]
Vintage	CensusSGM_r_128_4P_NoS	63.336 [ms]	15.7888 [fps]
Vintage	CensusSGM_r_128_4P_S	62.889 [ms]	15.901 [fps]
Vintage	CensusSGM_r_128_8P_NoS	68.095 [ms]	14.6854 [fps]
Vintage	CensusSGM_r_128_8P_S	62.927 [ms]	15.8914 [fps]
Vintage	CensusSGM_r_64_4P_NoS_C0	47.011 [ms]	21.2716 [fps]
Vintage	**CensusSGM_r_64_4P_NoS**	**39.959 [ms]**	**25.0257 [fps]**
Vintage	CensusSGM_r_64_4P_S	46.844 [ms]	21.3475 [fps]
Vintage	CensusSGM_r_64_8P_NoS	40.086 [ms]	24.9464 [fps]
Vintage	CensusSGM_r_64_8P_S	47.076 [ms]	21.2422 [fps]

TABLE VI: Time of Execution in milliseconds (ms) and FPS Benchmarks with the Middlebury Dataset of CensusSGM with different parameters , however all the above test use [P1=10, P2=120, Symmetric CENSUS] but only CENSUS C0 use the Classical CENSUS version. P = Number of Path e.g 4P = 4 Paths, S= SubPixelAccuracy, NoS= No SubPixelAccuracy used, f = Full Size images, r= Reduced size image, 128 and 64 are the size of the Disparity Map.