ARL-TN-0000 • Apr 2018

US Army Research Laboratory

GPU-Accelerated Symmetry Transform for Object Saliency

by Jason Owens

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

ARL-TN-0000 ● APR 2018

US Army Research Laboratory

GPU-Accelerated Symmetry Transform for Object Saliency

by Jason Owens
Vehicle Technology Directorate, ARL

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED (From - To)
April 2018	Technical Note	November 2016–April 2018

4. TITLE AND SUBTITLE
GPU-Accelerated Symmetry Transform for Object Saliency

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Jason Owens

5d. PROJECT NUMBER
APPLE

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
US Army Research Laboratory
ATTN: RDRL-VTA
Aberdeen Proving Ground, MD 21005-5066

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-TN-0000

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES
Primary author's email: <jason.l.owens.civ@mail.mil>.

14. ABSTRACT
This technical note describes a method to compute a symmetry transform in 2-D images, based on matching image gradients around a center reference point. The method is particularly amenable to parallel processing, and we describe an implementation that runs on graphics processing units for significantly reduced execution time for VGA-sized images. The symmetry transform has potential for use in finding salient regions containing objects, and also may be applicable to stable object keypoints for recognition.

15. SUBJECT TERMS
symmetry, parallel processing, object recognition, perception, intelligent systems

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Jason Owens
Unclassified	Unclassified	Unclassified	UU	24	19b. TELEPHONE NUMBER (Include area code) 410-278-7023

Contents

List of Figures

1. Introduction

A truly autonomous mobile agent needs to adapt and learn online in new environments. It simply not possible to provide all the information it will ever need ahead of time. In particular, an agent may encounter new objects it's never seen before. Therefore, the ability to detect these potentially novel objects is necessary for both recognition and continuous learning. While the larger task of continuous online learning is outside the scope of this particular note, it serves to define the context for an object saliency measure.

Finding salient object candidates in arbitrary natural images is a problem in computer vision and robotic perception that has yet to be solved. There are many ways this challenge is currently approached: 1) scan the entire image, at multiple scales, with some kind of object recognition algorithm, generating a heat map indicating the score or probability of a known, detected object (e.g., sliding window); 2) propose a smaller set of likely object regions based on some engineered or learned features of the image (e.g., edge boxes, geodesic object proposals, objectness); 3) compute a segmentation of the image to propose object-like regions; or 4) learn a saliency function to predict regions of the image that may contain objects.

In recent years and in combination with advanced deep learning systems for object recognition tasks, the use of some kind of object proposal algorithm has become a de facto standard. Since object proposal algorithms produce a significantly smaller number of image regions to test than almost any other mechanism (e.g., sliding window), they are used to speed up existing detection algorithms by reducing the number of times an object model must be checked. The main idea for many proposal algorithms is to either engineer or train a detector to respond to what has been called the "objectness" of a region (i.e., how well the pixels within the region correspond to the projection of an object on the image plane within the bounds of the region). Object aspects often include such properties as closed contours, convexity, and compactness.[1] In the image domain, the contours can often be reflected in the edges derived from the image gradient.

However, processing image contours without any additional information (e.g., learning an object contour prior) often yields undesirable results, with many proposals that do not represent an object. When we discuss objectness, we usually care about whole physical objects we can pick up (box of cereal, phone, pencil, flashlight, cof-

fee mug) and not the aspects of an object's appearance that may also provide strong gradient edge responses (e.g., logos or pictures on a cereal box).

In 1992, Reisfeld and Yeshurun proposed the use of a symmetry transform operator[2] in the image domain for use as an attentional operator. Since symmetry is considered a strong indicator of an object with shape,[3] it's reasonable to conclude that regions exhibiting strong symmetry are likely to contain objects of interest.

Reisfeld showed that by extracting contours from the symmetry magnitude and selecting local maxima, it was possible to compute attention keypoints that indicated important features of the image. For example, Fig. 1 shows an example from Reisfeld et al.[4] where selecting the maximum output of the radial symmetry transform produces keypoints for the face of Nixon and the forehead of Elvis, as well as several interesting points on the flags.

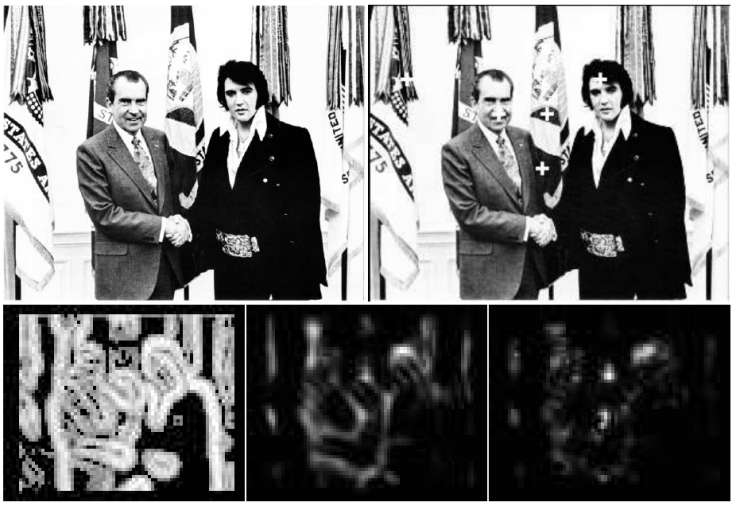

Fig. 1 Salient keypoint extraction using symmetry magnitude, reproduced from Reisfeld et al.[4] Reprinted by permission from: Springer Nature, International Journal of Computer Vision, Reisfeld et al., ©1995

Kootstra et al. have focused on the use of Gestalt features specifically for attention and segmentation of known objects,[5,6] exploiting and expanding on the concepts put forth by Reisfeld*. They found that while the symmetry informed the saliency and object proposal detection, their strong assumption that objects rest on (and protrude from) dominant planes proved the most useful at predicting object presence.

*Kootstra's work, in fact, was the inspiration for this investigation into implementing and parallelizing the symmetry transform for use in object saliency and proposal.

Potapova et al.[7] eschews the 2-D appearance-based symmetry and constructs a 3-D approach using depth maps derived from stereo or red, green, blue and depth (RGB-D) sensors, and compute symmetry by matching surface normal directions instead of gradient directions. The authors show this yields better objectness results than the appearance-based approach of Kootstra.

1.1 Problem

In general, we want to solve the ill-defined problem of finding regions in the image that contain objects or object parts. Objects are things in the world that are compact and have reasonably well-defined boundaries, and therefore will often be visible almost completely within an image frame. In contrast, "stuff" in the world is less well-bounded, and perhaps more amorphous and spread out (e.g., walls, carpet, and even desktops or counters in indoor environments).

In the theory of Gestalt perception, the whole is considered more than the sum of its parts.[8] Some of the phenomena that trigger this process include symmetry, continuity, and connectedness. In the context of this work, we are focusing on contours and accumulating the nature of their symmetry over the content of the image. The goal is to find regions associated with symmetric contours, and thereby derive some measure of the saliency of the region using the following observation: objects, by the nature of their shape and "well-boundedness," exhibit symmetry in their countours and boundaries. The rest of this note describes one method, and a specific implementation of it, for computing a transform that reflects the amount of symmetry surrounding a point, based on matching the direction of the gradients in the neighborhood of the point.

1.2 Contributions

In the following sections, we document our work implementing the symmetry algorithm directly from Reisfeld's mathematical formulation as an NVIDIA CUDA kernel for graphics processing unit (GPU) acceleration. We make the following contributions: a brief analysis of the computational implications of the algorithm, leading to the parallel nature of the problem; algorithm and source listing for the parallel kernel; and a brief discussion of the GPU performance and the output of a simple salient point detector.

2. Mathematical Approach

Reisfeld's symmetry transform uses the gradient image to compute symmetry magnitude and direction for every pixel in the image. Given the magnitude image, one can then use non-maximum suppression to select local maxima as salient points in order to guide the attention process. For convenience, we restate Reisfeld's mathematical formulation to provide background for the computational approach discussed in the next section.

Let $\mathcal{I} : \Omega \to [0, 1]$ be a grayscale image with domain $\Omega \subset \mathbb{Z}^2$. Then $p_k \in \Omega$ represents some pixel coordinate in the image, and $\nabla(p_k) = \left(\frac{\partial}{\partial x}\mathcal{I}(p_k), \frac{\partial}{\partial y}\mathcal{I}(p_k)\right)$ is the gradient at that coordinate. Reisfeld then computes a 2-D polar coordinate for each pixel, (r_k, θ_k), where $r_k = \log(1 + \|\nabla(p_k)\|)$, and $\theta_k = \operatorname{atan2}(\frac{\partial}{\partial y}\mathcal{I}(p_k), \frac{\partial}{\partial x}\mathcal{I}(p_k))$. Let l_{ij} be the line passing through 2 points p_i and p_j, and let α_{ij} be the angle l_{ij} makes with the horizontal (x) axis. For any pixel p_k, define the set $\Gamma(p_k) = \{(i, j) \mid \frac{p_i + p_j}{2} = p_k\}$, that is, the set of pixel index pairs such that p_k resides on the center of the line l. A distance, $D_\sigma(i, j)$, and phase, $P(i, j)$ function are used to determine the contribution $C(i, j)$ for each point pair in $\Gamma(p_k)$, defined as follows:

$$D_\sigma(i,j) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{\|p_i - p_j\|}{2\sigma}} \tag{1}$$

$$P(i,j) = (1 - \cos(\theta_i + \theta_j - 2\alpha_{ij}))\,(1 - \cos(\theta_i - \theta_j)) \tag{2}$$

$$C(i,j) = D_\sigma(i,j) P(i,j) r_i r_j \tag{3}$$

Finally, the *symmetry magnitude* for a point p is defined as:

$$M_\sigma(p) = \sum_{(i,j)\in\Gamma(p)} C(i,j) \tag{4}$$

which simply sums the weighted contributions over the entire "symmetric pixel" neighborhood of p (producing an averaged value). A direction contribution function for each pixel, $\psi(i, j)$, used to compute the *symmetry direction*, $\phi(p)$ for p, is defined as follows:

$$\psi(i,j) = \frac{\theta_i + \theta_j}{2} \tag{5}$$

$$\phi(p) = \psi(i^*, j^*) \quad \text{where} \quad (i^*, j^*) = \underset{(i,j)\in\Gamma(p)}{\operatorname{argmax}} C(i,j) \tag{6}$$

We can then define the overall output of the symmetry transform as:

$$S_\sigma(p) = \{M_\sigma(p), \phi(p)\}. \tag{7}$$

Please see Fig. 2 for an illustration indicating the values involved in the transform.

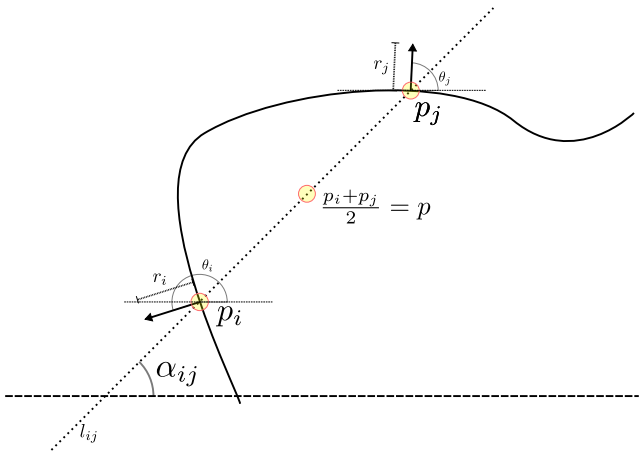

Fig. 2 Illustration of the geometry and quantities involved in computing pixel pair contributions for the image gradient symmetry around point p

Note that in the original paper, Reisfeld et al. justify using the logarithm of the gradient magnitude to reduce the contribution of stronger gradients and makes the correlation measure ($C(i,j)$) less sensitive to stronger edges.[4] Also note that the 2-D Gaussian function used in the distance function is circular; Reisfeld points out that this can be modified to emphasize elliptical features. In addition, he defines a modified symmetry magnitude he calls *radial symmetry* that emphasizes symmetries that are perpendicular to the primary symmetry direction (i.e., $\phi(p)$):

$$RS_\sigma(p) = \sum_{(i,j)\in\Gamma(p)} C(i,j)\sin^2\left(\psi(i,j) - \phi(i,j)\right). \tag{8}$$

An important aspect of this function is to note that the $M_\sigma(p)$ value must already be

computed (and $C(i, j)$ computed twice, or otherwise cached), by virtue of the use of $\phi(i, j)$, since it is a function of the entire neighborhood $\Gamma(p)$.

3. Computational Approach

From the mathematical definition of the symmetry transform, we can see that there are no mutual data dependencies between pixels given the gradient image; in other words, the problem is embarrassingly parallel. Each pixel *does* depend on a neighborhood (defined both by equation 1 and $\Gamma(\cdot)$), but $M_\sigma(p)$ does not need values computed by any other neighboring pixel. We can therefore compute the symmetry transform for each pixel independently, which suggests that an adaptation of the algorithm for GPU computation should be relatively straightforward.

In this section, we present the basic sequential algorithm (with no optimizations), and then discuss how it was readily adapted for computation on a GPU.

3.1 Sequential Algorithm

Algorithm 1 represents the pseudocode for the symmetry transform, which accepts the minimum radius σ, the gradient magnitude image g_m, and the gradient direction image g_θ, where $\sigma \in \mathbb{Z}$, $g_m : \Omega \to \mathbb{R}$ and $g_\theta : \Omega \to \mathbb{R}$.

Algorithm 1 Symmetry Transform

function SYMMETRY(σ, g_m, g_θ) ▷ Symmetry transform with radius σ for $\mathcal{I}$ with gradient magnitude g_m and gradient direction g_θ.
  $\rho \leftarrow 2.5\sigma$
  $S_m, S_\theta \leftarrow$ new arrays of $\mathbb{R}$ compatible with $\mathcal{I}$
  for $y \leftarrow 0, \text{rows}(\mathcal{I})$ **do** ▷ Iterate over all pixels in Ω
    for $x \leftarrow 0, \text{cols}(\mathcal{I})$ **do**
      $M, C_{ij}, \psi_{ij}, C_{\max}, \phi_p, \alpha_{ij} \leftarrow 0$
      $p \leftarrow [x, y]^\mathsf{T}$
      $p_{\min} \leftarrow p - \rho$
      $p_{\max} \leftarrow p + \rho$
      for $j \leftarrow p_{\min}[y], p_{\max}[y]$ **do** ▷ Iterate over all pixel indices in the square $(p_{\min}, p_{\max})$
        for $i \leftarrow p_{\min}[x], p_{\max}[x]$ **do**
          $p_i \leftarrow [i, j]^\mathsf{T}$
          $p_j \leftarrow p - (p_i - p)$ ▷ Compute the mirror point
          if $p_i = p$ **then**
            terminate neighborhood loop ▷ All remaining p_i, p_j pixel pairs are symmetric
          end if
          if $\text{valid\_pt}(p_i) \wedge \text{valid\_pt}(p_j)$ **then**
            $r_i, \theta_i \leftarrow \text{pt\_gradient}(g_m, g_\theta, p_i)$
            $r_j, \theta_j \leftarrow \text{pt\_gradient}(g_m, g_\theta, p_j)$
            $\delta_{ij} \leftarrow p_j - p_i$
            $\alpha_{ij} \leftarrow \text{atan2}(\delta_{ij}[y], \delta_{ij}[x])$
            $C_{ij} \leftarrow r_i r_j \, \text{D}(i, j, \sigma) \, \text{P}(\alpha_{ij}, i, j)$
            $M \leftarrow M + C_{ij}$
            $\psi_{ij} \leftarrow \frac{(\theta_i + \theta_j)}{2}$
            if $C_{ij} > C_{\max}$ **then**
              $C_{\max} \leftarrow C_{ij}$
              $\phi_p \leftarrow \psi_{ij}$
            end if
          end if
        end for
      end for
      $S_m(p) \leftarrow M$
      $S_\theta(p) \leftarrow \phi_p$
    end for
  end for
  return S_m, S_θ
end function

There are several specific aspects we highlight in this formulation. While Reisfeld leaves the definition of $\Gamma(p)$ open to *all* symmetric points surrounding p, the effects of points further away are clearly limited by the distance function $D_\sigma(i, j)$ (i.e., Eq. 1). In this code, we explicitly limit the bounds of the per-pixel neighbor iteration to a square region defined by the $p_{\min}$ and $p_{\max}$, with sides equal to $2\rho = 5\sigma$, see lines 10–11. This follows the implementation details given by Kootstra et al.,[5] where they use default minimum and maximum radii values of 7 and 17, respectively. In this implementation, σ is given as a parameter to the function to control the scale, and we compute the max radius as ρ directly from σ. Also, note that the argmax is implemented inline in lines 25–28 to avoid running another loop to compute the maximum. Finally, we only have to process *half* the neighborhood region, since by definition the other half of the points are symmetric to the first half; line 15 shows where this early termination occurs.

3.2 Parallel Adaptation

To adapt Alg. 1 for the GPU, we simply extract lines 6–33 and convert them to an appropriate GPU kernel function (see the code listing in Alg. 2). In our implementation, we use the CUDA language for NVIDIA GPUs (the most common discrete GPU in our environment). We do not claim any particular ingenuity in converting this problem to a parallel implementation; we simply document the implementation and show the performance benefits.

Algorithm 2 Parallel Cuda Kernel

```
1  __global__ void cu_symmetry(cv::gpu::PtrStepSzf smag,
2                              cv::gpu::PtrStepSzf sdir,
3                              const cv::gpu::PtrStepSzf mag,
4                              const cv::gpu::PtrStepSzf dir,
5                              int sigma)
6  {
7    // prepare region for given pixel x,y
8    int x = blockIdx.x * blockDim.x + threadIdx.x;
9    int y = blockIdx.y * blockDim.y + threadIdx.y;
10   int radius = (int)(2.5 * sigma);
11   int ROWS = mag.rows;
12   int COLS = mag.cols;
13
14   // target point, we are computing the symmetry for THIS point
15   int2 p = make_int2(x, y);
16   if (p.x >= COLS || p.y >= ROWS) return;
17
18   int2 min = make_int2(p.x - radius, p.y - radius);
19   int2 max = make_int2(p.x + radius + 1, p.y + radius + 1);
20   int2 pi, pj, d;
21   float2 rtheta_i, rtheta_j;
22   float M = 0;
23   float C_ij = 0;
24   float psi_ij = 0;
25   float maxC = 0;
26   float maxTheta = 0;
27   float alpha_ij = 0;
28   for (int j = min.y; j <= p.y; j++) {
29     for (int i = min.x; i < max.x; i++) {
30       if (abs(i-p.x) < sigma &&
31           abs(j-p.y) < sigma) continue;
32       pi = make_int2(i,j);
33       if (pi == p) break; // we are done, since this computation is symmetric
34       d = pi - p;
35       pj = p - d;
36       if (valid_pt(pi, ROWS, COLS) && valid_pt(pj, ROWS, COLS)) {
37         rtheta_i = pt_gradient(mag, dir, pi);
38         rtheta_j = pt_gradient(mag, dir, pj);
39         alpha_ij = atan2f(pi.y - pj.y, pi.x - pj.x);
40         C_ij = rtheta_i.x * rtheta_j.x * dist(pi, pj, (float)sigma) *
41           phase(alpha_ij, rtheta_i.y, rtheta_j.y);
42         M += C_ij;
43         psi_ij = (rtheta_i.y + rtheta_j.y) * 0.5;
44         if (C_ij > maxC) {
45           maxC = C_ij;
46           maxTheta = psi_ij;
47         }
48       }
49     }
50   }
51   smag(p.y,p.x) = M;
52   sdir(p.y,p.x) = maxTheta;
53 }
```

Note that it is impossible to get full utilization of the thread warps near the edges of the image, since some threads will be sitting idle due to the `valid_pt` checks ensuring we don't process pixels outside the image bounds. Also note in lines 30–31 that we apply the modification from Kootstra et al.[5] that ignores the gradients near the center of the point, to help emphasize the gradients at the given radius

(σ); this produces a "no computation zone" that causes portions of thread warps to become idle, due to the single-instruction multiple-data (SIMD) nature of the CUDA computation model.

The kernel is then called with a default block size of 16×16. No optimizations have yet been implemented for device-dependent occupancies or shared memory usage. Both adjustments could improve overall performance, but require additional complexity in the kernel and the host calling function.

4. Results

4.1 Performance

The performance difference between single central processing unit (CPU) and GPU implementations is staggering. As Fig. 3 shows using a logarithmic-scale y axis, there are 2 orders of magnitude improvement in execution time for the GPU version, primarily due to the massive parallelism on the GPU and the virtually nonexistent dependencies between pixel values (i.e., no reductions necessary, and no waiting), even with idle threads due to the internal "no computation zone."

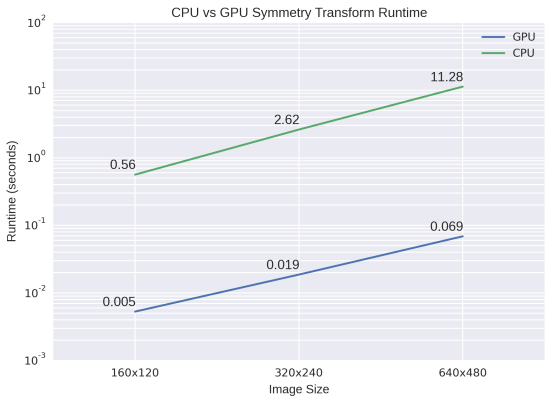

Fig. 3 Single CPU vs. GPU performance comparison. Note that there are 2 orders of magnitude improvement between the GPU and the CPU runtime. Tests were performed on an Intel 4th-Gen Core i7 2.9 GHz processor and NVidia Quadro M3000M GPU with 768 CUDA cores

4.2 Transform Output

For our purposes, we are much less interested in the symmetry direction as we are in the symmetry magnitude (as discussed in the introduction). Figure 4 shows example output from the implemented symmetry transform as well as simple "fea-

ture" detection output derived from the symmetry magnitude images. A quick and dirty method for finding features is to use non-maximum suppression to find local maxima (and then suppress any other maxima within a given radius). To show the usefulness of the transform on natural images, we implemented a simple detector that computes an image pyramid, runs the symmetry transform on each layer, and then merges the result into a single full-scale magnitude image in order to find local maxima. In the results shown, our detector uses a suppression radius of 15 pixels.

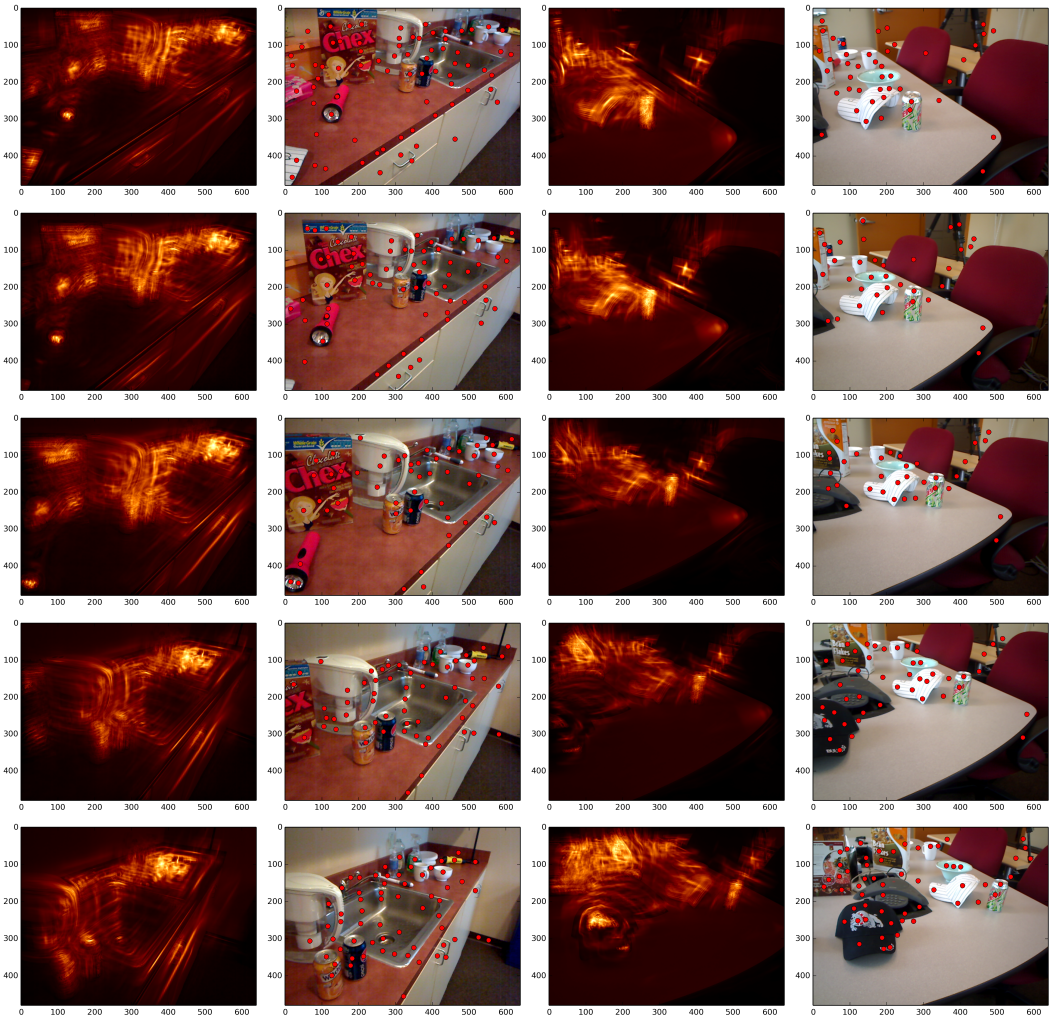

Fig. 4 Five consecutive images each of the `kitchen_small` (left pair) and `meeting_small` (right pair) scenes from the `rgbd_scenes` data set.[9] For each pair, the computed symmetry magnitude is shown on the left, and the local maxima found using non-maximum suppression is shown on the right, using radius 15

The images show significant detection coverage in the image regions with objects, although there are keypoints found in “stuff” regions (e.g., the front of the kitchen

counter), and the keypoints are not stable across multiple frames. In particular, while points fall within the object boundaries, due to the appearance of the object and the camera viewpoint, some keypoints change position over time. Therefore, this methodology should not be used to extract precision keypoints for image matching, and any use for extracting descriptors or image regions for object recognition should be robust to at least small variations in the position of the region centroid.

5. Future Work

While Reisfeld[4] and Kootstra's[5,6,10,11] work (and therefore this work) focused on 2-D appearance, and Potapova et al. focused on 3-D point cloud symmetry, we believe the next best step is to consider some combination of the 2 approaches that exploits the capabilities of each modality. For example, by combining approaches, we may be able to detect posters on walls (an object without detectable depth) as well as remove the spurious appearance-based detections on the kitchen counter by checking for plane membership over extended bounds.

6. Conclusion

This work documents the initial steps in a research program to find an object saliency algorithm for use in a continuous object discovery and learning system. We reviewed the mathematical approach to the problem, described the computational approach, and created a parallel kernel suitable for acceleration on a GPU. We show the significant performance improvement through parallelization, and discuss the qualitative results from a simple keypoint detector.

7. References

1. Koffka K. Principles of Gestalt psychology. New York: Harcourt, Brace, & World; 1935. OCLC: 248396251.

2. Reisfeld D, Yeshurun Y. Robust detection of facial features by generalized symmetry. In: Robust detection of facial features by generalized symmetry; IEEE Comput. Soc. Press; 1992. p. 117–120.

3. Li Y, Sawada T, Shi Y, Steinman RM, Pizlo Z. Symmetry is the sine qua non of shape. In: Shape Perception in Human and Computer Vision; Dickinson SJ, Pizlo Z, editors. London: Springer London; 2013; p. 21–40.

4. Reisfeld D, Wolfson H, Yeshurun Y. Context free Attentional operators: The generalized symmetry transform. Int. J. Comput. Vision. 1995;14:119–130.

5. Kootstra G, Bergstrom N, Kragic D. Using symmetry to select fixation points for segmentation. In: Pattern Recognition (ICPR), 2010 20th International Conference on; IEEE; 2010. p. 3894–3897.

6. Kootstra G, Bergström N, Kragic D. Gestalt principles for attention and segmentation in natural and artificial vision systems. In: ICRA 2011 Workshop on Semantic Perception, Mapping and Exploration (SPME), Shanghai, China; eSMCs; 2011.

7. Potapova E, Zillich M, Vincze M. Local 3d symmetry for visual saliency in 2.5 d point clouds. In: Asian Conference on Computer Vision; Springer; 2012. p. 434–445.

8. Jäkel F, Singh M, Wichmann FA, Herzog MH. An overview of quantitative approaches in Gestalt perception. Vision Research. 2016;126:3–8.

9. Henry P, Krainin M, Herbst E, Ren X, Fox D. RGB-D mapping: Using Kinect-style depth cameras for dense 3D modeling of indoor environments. The International Journal of Robotics Research. 2012;31(5):647–663; bibtex: Henry2012.

10. Kootstra G, Bergström N, Kragic D. Fast and automatic detection and segmentation of unknown objects. In: 2010 10th IEEE-RAS International Conference on Humanoid Robots; IEEE; 2010. p. 442–447.

11. Kootstra G, Kragic D. Fast and bottom-up object detection, segmentation, and evaluation using Gestalt principles. In: Robotics and Automation (ICRA), 2011 IEEE International Conference on; IEEE; 2011. p. 3423–3428.

List of Symbols, Acronyms, and Abbreviations

2-D	2-dimensional
3-D	3-dimensional
CPU	central processing unit
GPU	graphics processing unit
RGB-D	red, green, blue and depth
SIMD	single-instruction multiple-data
VGA	Video Graphics Array

INTENTIONALLY LEFT BLANK.

1 (PDF) DEFENSE TECHNICAL INFORMATION CTR
DTIC OCA

2 (PDF) DIR ARL
IMAL HRA
RECORDS MGMT
RDRL DCL
TECH LIB

1 (PDF) GOVT PRINTG OFC
A MALHOTRA

1 (PDF) UNIVERSITY OF PENNSYLVANIA
DEPT COMPUTER SCIENCE
K DANIILIDIS
kostas@cis.upenn.edu

ABERDEEN PROVING GROUND

8 (PDF) DIR USARL
RDRL VT
A GHOSHAL
B SADLER
RDRL VTA
MA FIELDS
H EDGE
C KRONINGER
J OWENS
P OSTEEN
RDRL HRF-D
T KELLEY

INTENTIONALLY LEFT BLANK.