Chantal Ding Chloé Macur

December 2013

Symmetry Detection

A Planar-Reflective Symmetry Transform for 3D Shapes

Project report
INF555 – Digital Representation and Analysis of Shapes

Contents

Introduction

A lot of 3D shapes, whether natural or man-made, present some kind of symmetry that can be really useful in computer vision and 3D geometry. Indeed symmetries allow certain economy, especially in digital representation, but they are also involved in pattern recognition or geometry completion. Thus, numerous methods are used to detect symmetries, partly because of the diversity of datas (point clouds, polygon meshes, NURBS, patches, etc.).

1 A Planar-Reflective Symmetry Transform for 3D Shapes

We chose to try to implement the planar reflective symmetry transform (PRST) described by Podolak et al. [2]. This transform takes a 3D shape and aims at getting the space of planes that are associated to a reflectional symmetry. It provides both global symmetries, certain local symmetries and even imperfect symmetries. Moreover, it is not sensitive to noise or missing data. For a proper understanding of this method, we transcribe here some of the formulas used.

Given a function f representing the energy, $PRST(f,\gamma)$ is a measure of f's symmetry with respect to the plane reflection γ. The symmetry distance, $SD(f,\gamma)$ is defined as the L_2 distance between f and the nearest function that is invariant to that reflection:

$$SD(f,\gamma) = min_{g|\gamma(g)=g}\|f-g\|.$$

Which leads, by normalizing the symmetry measure, to

$$PRST^2(f,\gamma) = 1 - \frac{SD^2(f,\gamma)}{\|f\|^2}$$

However, the nearest symmetric function to f is the average of f and $\gamma(f)$:

$$SD(f,\gamma) = \|f - \frac{f+\gamma(f)}{2}\| = \frac{\|f-\gamma(f)\|}{2}$$

Therefore, we have:

$$PRST^2(f,\gamma) = 1 - \frac{SD^2(f,\gamma)}{\|f\|^2} = 1 - \frac{\|f-\gamma(f)\|^2}{4\|f\|^2} = 1 - \frac{\|f\|^2 - 2f.\gamma(f) + \|\gamma(f)\|^2}{4\|f\|^2}$$

If f is normalized, we obtain

$$PRST^2(f,\gamma) = \frac{1+f.\gamma(f)}{2}$$

Thus, to evaluate the symmetry measure for a plane, we need to evaluate the following estimator, called Monte Carlo estimator :

$$D(f,\gamma) = f.\gamma(f).$$

To compute this PRST we chose to use a Monte Carlo algorithm, which computes a discrete version of the PRST that takes advantage of sparsity in the function f. This sparsity is mostly found in point sets and rasterized surfaces. The brute force approach would be to evaluate the PRST for every single plane in the space of the figure. That would be done by selecting a pair of points and voting for the plane between them. Instead of chosing the points randomly, we favour the points corresponding to a high energy of function f.

for sampled points x:
  for sampled points x':
    $\gamma \leftarrow$ reflection plane(x,x')
    $D(f,\gamma) += w(x,x',\gamma).f(x).f(x')$

Moreover we do not vote for planes but discretized bins of planes and we weight the vote of each pair of points:

$$w(x,x',\gamma) = w_{samp}.w_{change-of-variables}$$

w_{samp} is the reciprocal of the probability of having selected the two points :

$$w_{samp}(x, x', \gamma) = \frac{1}{f(x).f(x')}.$$

The $change - of - variables$ is due to the fact that we move from a pair of points to discretized bins represented by polar coordinates. In 3D spherical coordinates, it is given by the formula

$$w_{change-of-variables} = 2d^2 \sin\theta$$

where θ is the angle of the bin from the origin.

This leads to

$$w(x, x', \gamma) = \frac{1}{f(x)f(x')2d^2 \sin\theta}.$$

In the end, we obtain

$$D(f, \gamma) = \frac{1}{N_{samp}} \sum_{i=1}^{N_{samp}} \frac{1}{2d^2 \sin\theta}$$

2 Our implementation

2.1 Conventions and Notations

We chose to use classic image formats (png, jpg) as input files as they are naturally rasterized. For more simplicity we only considered grayscale images. Since the method we are implementing is taking advantage of sparsity in the function, a typical input image would be a black and white outline (see Figure 1). The Java class `BufferedImage` provides an easy way to access such images and manipulate them as arrays of integers between 0 and 255.

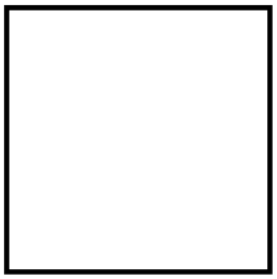


Figure 1: Example of input image

Since pixel values are high for white and low for black, we want to work with the function f defined as follows for each pixel p of the image I :

$$f(p) = 255 - \text{intensity of } I \text{ at } p$$

so that $f(p) = 0$ for white pixels, i.e. empty space.

For point coordinates, we simply use the default Java coordinate system, with $(0, 0)$ in the upper-left corner, as shown in Figure 2. All points have integer coordinates. Lines however, are parametrized over a pair of `double` values (r, θ) where $|r|$ is the distance of the line to the origin and θ the angle of the normal vector, as shown in Figure 3. In order to easily retrieve θ from $\sin\theta$, we chose to have $\theta \in \left]-\frac{\pi}{2}, \frac{\pi}{2}\right]$ and the radius r can be negative.

2.2 Sampling

In order to compute the PRST using the Monte Carlo algorithm for sparse functions, we first perform an importance sampling of points in the image, using $f(x)$ as the probability density of selecting x.

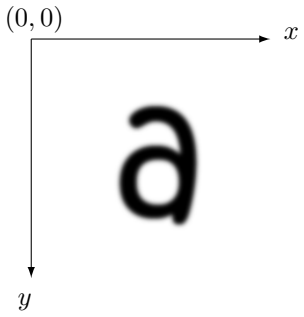


Figure 2: (x, y) coordinates for Java images

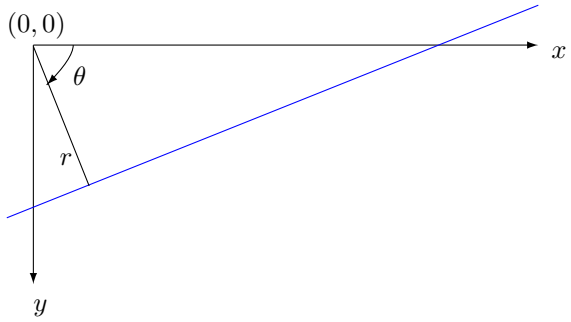


Figure 3: (r, θ) parametrization of lines

For this, we use the cumulative sum of $f(x)$ over the image as the cumulative distribution function and perform an inverse transform sampling. Usually, this method works by generating a real number u in range [0,1] and computing $F^{-1}(u)$, where F is a cumulative distribution function. Here, we adapted the method to work with non-normalized integer functions :

1. Generate a random integer n in range $[0, max_F]$, where F is the cumulative sum of pixel values and max_F its maximum value;
2. Let $p_1, ..., p_N$ be the pixels of the image, compute $i = \inf\{m \in \mathbb{N}, m \leq N | F(p_m) = n\}$;
3. Add p_m to the sample.

2.3 Adjustments for 2D

The Monte Carlo algorithm we chose to use for computing the PRST requires to derive a change-of-variable weight for each pair of points (x, x'), accounting for the transformation between the discrete parametrization of lines over (r, θ) and the pairs of reflected points. In 3D, the change-of-variables weight is given by the following formula :

$$w_{change-of-variables} = 2d^2 \sin\theta$$

However, since we are working in 2D, we need to recompute this weight in order to fit the new coordinates system. As in 3D, we compute the determinant of the Jacobian of the change-of-variables transformation. Let

$$\vec{n} = \begin{pmatrix} \cos\theta \\ \sin\theta \end{pmatrix}$$

be the normal of the axis of reflection, d the distance between x and x', we can write :

$$\begin{aligned} d &= \|x - x'\| \\ &= 2(r - \vec{n} \cdot x) \\ x' &= x + d\vec{n} \\ &= x + 2(r\vec{n} - (\vec{n} \cdot x)\vec{n}) \end{aligned}$$

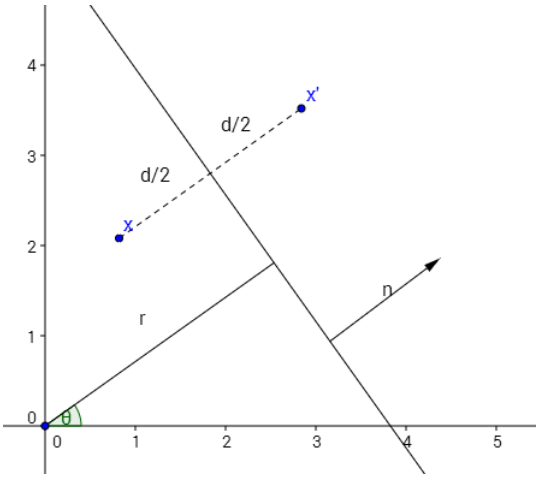


Figure 4: cartesian and polar coordinates

thus we have :

$$\frac{\partial x'}{\partial r} = 2\vec{n}$$
$$\frac{\partial x'}{\partial \theta} = 2r\vec{u} - 2[(\vec{u}\cdot x)\vec{n} + (\vec{n}\cdot x)\vec{u}]$$

where

$$\vec{u} = \frac{\partial \vec{n}}{\partial \theta} = \vec{n} = \begin{pmatrix} -\sin\theta \\ \cos\theta \end{pmatrix}.$$

Therefore, the Jacobian is

$$J = 2\begin{pmatrix} \cos\theta & -r\sin\theta - (\vec{u}\cdot x)\cos\theta + (\vec{n}\cdot x)\sin\theta \\ \sin\theta & r\cos\theta - (\vec{u}\cdot x)\sin\theta - (\vec{n}\cdot x)\cos\theta \end{pmatrix}$$

and so the determinant is

$$w_{change-of-variables} = |J| = 2d.$$

We were initially suprised by the lack of dependence in θ, contrary to the 3D formula given in the paper. However, noting that the cartesian-to-polar (2D) and cartesian-to-spherical(3D) change-of-variables jacobian determinants are respectively given by

$$|J_{cartesian-to-polar}| = r = \frac{w_{2D-change-of-variables}}{2}$$

and

$$|J_{cartesian-to-spherical}| = r^2\sin\theta = \frac{w_{3D-change-of-variables}}{2}$$

gives a good intuition on why there is such a descrepancy between the 2D and 3D weights. Finally, the 2D Monte Carlo estimator is given by :

$$D(f,\gamma) = \frac{1}{N_{samp}}\sum_{i=1}^{N_{samp}}\frac{1}{2d}.$$

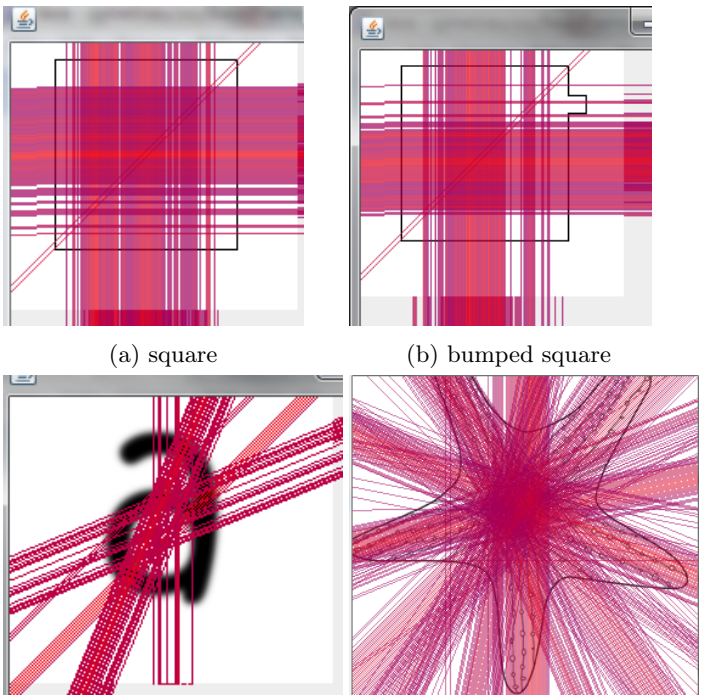
2.4 Comments

Choice of the 1/N factor in the calculus of D : explain that it comes out of the blue ...

Bounds : la valeur du prst (monte carlo) obtenue telle qu'on le fait dépend complètement de l'unité de longueur qu'on utilise pour calculer les distance x,x'. Du coup on n'a vraiment aucune garantie sur les bornes du prst Monte Carlo, contrairement au prst complet...

3 Results

We visualized the computed PRST applying a treashold proportional to the maximum PRST value, and tracing the lines corresponding to the strongest symmetries. The lines are traced in colors ranging from blue to red, with red for the stronger symmetries and blue for the weaker ones. We used an alpha channel to add transparency to the lines and allow the original image to show through, as well as making areas with high density of lines more legible. Figure 5 shows a few examples of computed PRST.



(a) square (b) bumped square

(c) the letter a (d) a starfish

Figure 5: Visualizations of the PRST for several 2D outlines

One of the main claims of PRST is its stability and insensibility to noise. We empirically tested the claim by comparing the PRST visualization of a square and a modified version of the square with a small bump added to the outline. As can be seen in Figure 5a and Figure 5b, the two inputs yield similar results. The PRST is also good at detecting approximate symetries, and we had fairly good results with the image of a starfish (Figure 5d).

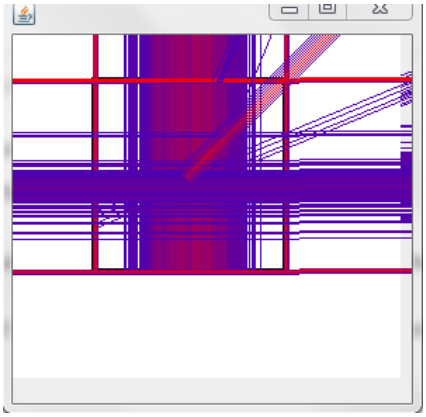


Figure 6: Example of "overbearing" local symetries

However, our implementation presents a few weaknesses. Some local symetries appear with too much emphasis, in particular for input images containing thick straight lines. Figure 6 gives a good example

of the phenomenon : the local symetries of the four sides of the squares appear to have a higher PRST than the main axis of symetry. We believe it is due to the $\frac{1}{2d}$ term in $D(f, \gamma)$ giving too much importance to local symetries in dense areas. The phenomenon persists even with a sampling size as high as 10,000 points, suggesting that sampling size is unable to compensate for the weight given to closer pairs of points.

The threashold method for visualizing the PRST is also imperfect. In order to obtain legible results, we need to tweak the threashold value for every change of input image or sample size. Despite the $\frac{1}{N_{samp}}$ normalization, there is no "universal" threashold one can use to discriminate strong symetries from weak symetries in an image. For input images containing numerous axes of symetry, this results in difficult to read visualization (see Figure 7, since a high number of axes will have a PRST value close to the maximum. Figure 8 shows that using a higher threashold does not give better results.

Figure 7: Visualization of PRST values of an image with a high number of symmetries

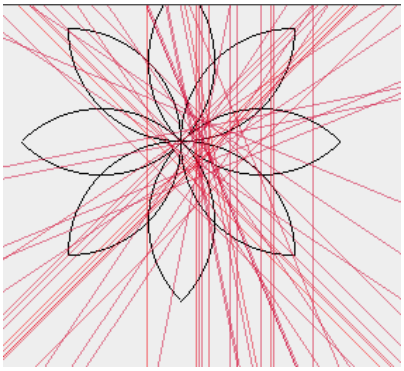


Figure 8: Visualization of PRST values of an image with a high number of symmetries, with a threashold of $0.8max$

Conclusion

We have implemented the Monte Carlo algorithm to calculate a discrete version of the planar reflective symmetry transform for 2D shapes, by adapting the 3D method in 2D. We managed to perform an efficient sampling according to the density of points in the image. Then we selected pairs of sampled points and voted for the most accurate bin of planes between them. We managed to obtain good results namely high PRST values corresponding to symmetry axes in the original picture. As a result, we measured the symmetry of an object with respect to all planes in a discrete space. Computer graphics benefit from that type of symmetry information, which allows a certain efficiency and economy in shape processing.

References

[1] Niloy J. Mitra, Mark Pauly, Michael Wand, and Duygu Ceylan. Symmetry in 3d geometry: Extraction and applications. In *EUROGRAPHICS State-of-the-art Report*, 2012.

[2] Joshua Podolak, Philip Shilane, Aleksey Golovinskiy, Szymon Rusinkiewicz, and Thomas Funkhouser. A planar-reflective symmetry transform for 3D shapes. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 25(3), July 2006.