15th IAPR International Conference on Machine Vision Applications (MVA)
Nagoya University, Nagoya, Japan, May 8-12, 2017.

Detection of Self Intersection in Synthetic Hand Pose Generators

Shome Subhra Das
CVAI lab, Electrical Engineering Department
Indian Institute of Science, Bangalore
shome@ee.iisc.ernet.in

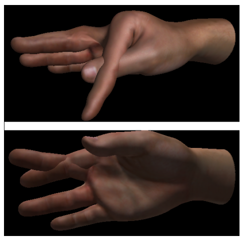
Abstract

Synthetic hand pose data has been frequently used in vision based hand gesture recognition. However existing synthetic hand pose generators are not able to detect intersection between various hand parts and can synthesize self intersecting poses. Using such data may lead to learning wrong models. We propose a method to eliminate self intersecting synthetic hand poses by accurately detecting intersections between various hand parts. We model each hand part as a convex hull and calculate pairwise distance between the parts, labeling any pair with a negative distance as intersecting. A hand pose with at least one pair of intersecting parts is labeled as self intersecting. We show experimentally that our method is very accurate and performs better than existing techniques. We also show that it is fast enough for offline data generation.

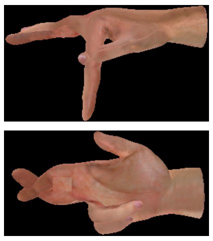
1 Introduction

Vision based hand gesture recognition methods extensively use machine learning algorithms which require high volumes of training data. Capturing such data manually is a very tedious job. Hence synthetic (rendered) data has been used in hand gesture related research [9, 10, 11, 17, 20, 21, 22, 14].

Linear blend skinning (LBS) is a popular technique for rendering articulated objects. Currently there exists two synthetic hand generators namely Libhand [15] and Handgenerator[7] that can generate RGB and depth images of all possible hand poses. Both use LBS to model the human hand. However both hand renderers suffer from a limitation that there is no automatic and accurate way to disable the generation of self intersecting hand poses. Fig. 1a and Fig. 1b show examples of self intersecting hand poses synthesized by LibHand and HandGenerator respectively.



(a) By LibHand



(b) By HandGenerator

Figure 1: Self intersecting hand pose samples

In this paper we propose a method to accurately detect self intersections in synthetic hand pose data generated using LBS technique. We model all hand parts using convex hulls and find the penetration depth between these convex hulls using GJK-EPA algorithm ([4], [23]). We label pairs of hand parts as intersecting if they have negative penetration depth. We show experimentally that our method is very accurate and better than existing techniques. We also show that our method is fast enough for offline data generation.

2 Related Work

There exist three broad methods to eliminate self intersecting poses:

- By imposing static and dynamic hand joint angle constraints based on anatomical studies.
- Using geometric primitives to represent fingers and 3D geometry to compute inter finger overlap.
- Using Rigid body dynamics with pairwise constraints between hand parts to avoid self intersection.

Algorithm	Method	Application
Oikonomidis et al. [10], Schroder et al. [16]	JA	Hand Tracking
Liang et al. [12]	JA	Hand Parsing
Choi et al. [13], Zhou et al. [24]	JA	Hand Pose estimation
Supancic III et al. [18]	JA	Hand Pose data creation
Tompson et al. [22]	GP	Hand Pose Estimation
Tagliasacchi et al. [25]	GP	Hand Tracking
Melax et al. [6]	RBD	Hand Tracking
Krejov et al. [5]	RBD	Hand Pose Estimation

Table 1: Existing work to remove self intersecting poses. JA: Joint angle based, GP: Geometrical Primitives based, RBD: Rigid body dynamics based.

Joint angle constraints hold true for average human hand and are not accurate for every hand shape. Methods using simplified geometrical models do not accurately model the hand and hence their calculations are not accurate either. Rigid body dynamics along with constraints on velocity, impact forces has been used for avoiding self intersection. However it is not suitable for data generation as multiple sequential frames are needed for velocity, force computations.

We show experimentally that the proposed method works very accurately in all scenarious including all cases where existing methods fail.

3 Prerequisite Concepts

How Synthetic Hand pose generators Work: Synthetic Hand pose generating softwares (Fig. 2) take

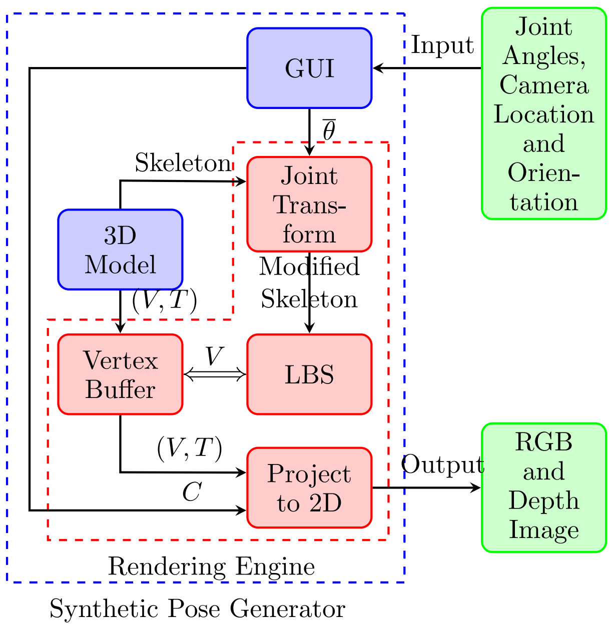


Figure 2: General schematic of Synthetic Pose generator: V = Vertex locations, T = Texture Coordinates, C = Camera Location and Orientation, θ = Joint angles.

hand joint angles along with the location and orientation of a virtual camera as input from a GUI and render synthetic RGB and depth images of various hand poses. A 3D hand model consisting of mesh vertices, a skeleton and a skin texture image (Fig. 3a) is loaded into the rendering engine. The 3D vertex locations and texture coordinates are stored in the vertex buffer. By applying LBS technique, the 3D vertex locations of the mesh are modified to match the current skeleton configuration as set by the input joint angles. The projection matrix is applied on the modified vertex locations to get the 2D rendered image. The color on the image is set using the foreground vertex colors from the texture image taking care of the lighting conditions.

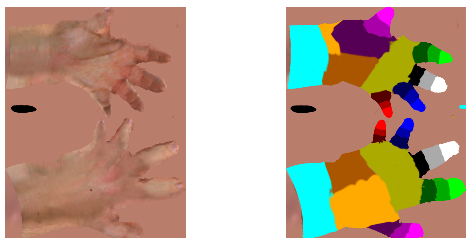
Finding Distance Between Convex Hulls: Boolean GJK algorithm [8] detects if two convex hulls intersect. Expanding Polytope Algorithm (EPA) [23] finds the distance between two intersecting convex hulls. GJK along with EPA is often used in computer game design for collision detection.

4 Proposed Method

We propose a method to accurately detect intersections between various hand parts of a synthesized hand pose. The hand mesh and the segmented texture image (Fig. 3b) are loaded into the rendering engine (Fig. 2). From the vertex buffer of the rendering engine we extract the 3D location of the vertices (V) and the corresponding texture coordinates (T) after the locations of vertices have been modified according to the input joint angles (using LBS). We segment the vertices using color label corresponding to each part and find the convex hulls for all the segmented hand parts (Fig. 4). The penetration depth between these convex hulls are calculated using GJK-EPA algorithm. We label pairs of hand parts as intersecting if they have negative penetration depth. The penetration depth is thresholded to allow small compression of hand parts. A hand pose is declared as self intersecting if any two non adjacent hand parts intersect each other. Once self-intersection is ruled out, the skin colored texture image is used to generate hand pose data.

Algorithm 1 Self Intersection Detection algorithm

```
1: procedure DETECT SELF INTERSECTION
2:     Extract vertex locations from vertex buffer.
3:     Assign color to each vertex by indexing the texture image with the texture coordinates.
4:     Segment the vertices based on color.
5:     Fit Convex hulls to all sets of segmented vertices.
6:     Using GJK-EPA algorithm compute pairwise distance between all convex hulls (hand parts).
7:     Label any pair as intersecting if the distance between them is negative and lesser than a threshold
8:     Label the hand pose as self intersecting if there is any pair of intersecting parts
9: end procedure
```



(a) Skin Texture (b) Segmented Texture

Figure 3: Texture Images

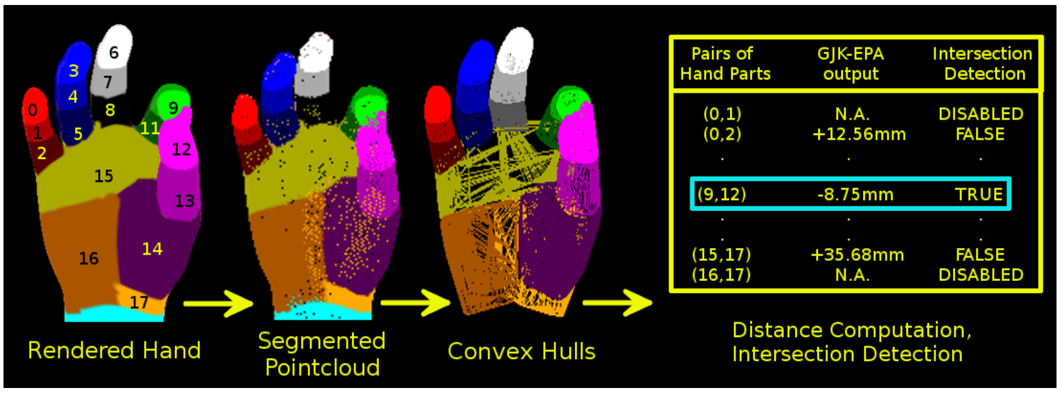
5 Experimental Results:

Setup: We used the Hand renderer code at `https://github.com/jsupancic/libhand-public`. We modified the code to extract the vertices and texture coordinates from the vertex buffer. Convex hull computation and pairwise distance calculation (using GJK-EPA) were done using the Bullet Physics Library [3]. We use a threshold of 2 mm on the penetration depth to allow for small compressions of the hand parts. Nerve endings in the finger are found at a depth of 0.7 mm to 1.0 mm below the skin surface ([19]). So 2 mm is the maximum penetration before finger-finger intersection is perceivable.

Results: We captured various intersecting poses by varying the number of intersecting hand parts and the level of penetration from shallow to deep. Table 3 shows that the detection of self intersections by our method was accurate in absolute terms in all cases.

To compare with joint angle method we generate test cases which satisfy the limits from [2] but result to self intersections. Such cases are not detectable by the joint angle method. Table 4 shows that our method performs accurately with all the intersection cases that were not detected by the joint angle limit method.

To compare with methods using geometric primitives we create test cases which are very close to self intersection but are not intersecting (as in Fig. 5). We calculated the penetration as specified in [22] and



Pairs of Hand Parts	GJK-EPA output	Intersection Detection
(0,1)	N.A.	DISABLED
(0,2)	+12.56mm	FALSE
.	.	.
(9,12)	-8.75mm	TRUE
.	.	.
.	.	.
(15,17)	+35.68mm	FALSE
(16,17)	N.A.	DISABLED

Figure 4: Dataflow along our pipeline: The first image is the hand rendered by LibHand, in which the tip of the thumb and the forefinger intersect. The second image represents the segmented pointcloud while the third image displays the convex hulls of the segmented parts. The table at the end shows the pairwise distance calculation and intersection detection using GJK-EPA algorithm. The hand parts are numbered from 0 to 17 as shown.

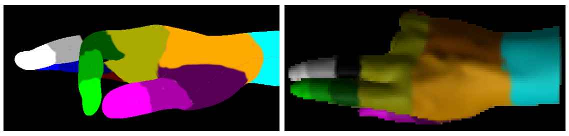

(a) Hand Pose 1 (b) Hand Pose 2

Figure 5: Samples of Non Intersecting test cases for comparison with Geometric primitive based methods.

Method	Part Model	Penetration Fig. 5a	Penetration Fig. 5b
[22]	Sphere	2500 mm^3	270 mm^3
[25]	Cylinder	632 mm^2	3 mm^2
Ours	LBS	No penetration	No penetration

Table 2: Poses (ref Fig.5a and Fig.5b) where methods based on geometric primitive fail while our's works as expected.

[25]. Table 2 shows that methods based on geometric primitives give incorrect penetration detections in both cases while our method accurately determines them as non intersecting.

Comparison with Rigid body dynamics based methods is not applicable as they require multiple sequential poses to compute velocity, force while data generation process needs each pose data to be independent of the rest.

On an i7-3770K PC our method took 200 ms to process a single frame (equivalent to 10^5 samples in 6 hrs) which is good enough for offline data generation.

6 Conclusion

We have proposed and experimentally verified a method to detect self intersecting hand poses created by synthetic hand pose generators. To find intersection between hand parts we construct convex hulls for all hand parts and calculate distance between the same using GJK-EPA algorithm. Pairs of parts which have negative distance are said to be intersecting. A pose which has at least one pair of intersecting hand parts is considered as self intersecting and is eliminated. We have demonstrated experimentally that our method works well in all intersection scenarios and is suitable for offline data generation. We have analyzed failure cases of existing methods and showed our method's accuracy in such cases. Our method is applicable to all synthetic pose generation softwares using LBS and will benefit researchers who design or use the same.

References

[1] G. v. d. Bergen. A fast and robust gjk implementation for collision detection of convex objects. *Journal of graphics tools*, vol.4, no.2, pp.7–25, 1999.

[2] S. Cobos et al. Efficient human hand kinematics for manipulation tasks. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp.2246–2251, 2008.

[3] E. Coumans et al. Bullet physics library. *Open source: bulletphysics. org*, 15, 2013.

[4] E. G. Gilbert et al. A fast procedure for computing the distance between complex objects in three-dimensional space. *Robotics and Automation, IEEE Journal of*, vol.4, no.2, pp.193–203, 1988.

[5] P. Krejov et al. Combining discriminative and model based approaches for hand pose estimation. In *11th IEEE International Conference and Workshops onAutomatic Face and Gesture Recognition*, vol.1, pp.1–7, 2015.

[6] S. Melax et al. Dynamics based 3d skeletal hand tracking. In *Proc. of Graphics Interface 2013*, pp.63–70. Canadian Information Processing Society, 2013.

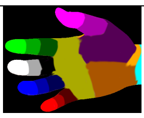
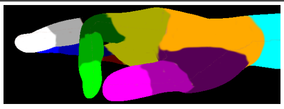
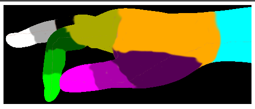
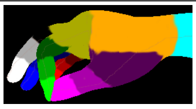
[7] A. Memo et al. Exploiting silhouette descriptors and synthetic data for hand gesture recognition. Citeseer 2015.

[8] C. Muratori. Implementing gjk, 2006.

[9] N. Neverova et al. Hand segmentation with structured convolutional learning. In *Asian Conference on Computer Vision*, pp.687–702. Springer, 2015.

[10] I. Oikonomidis et al. Efficient model-based 3d tracking of hand articulations using kinect. In *BMVC*, vol.1, no.2, pp.3, 2011.

[11] G. Rogez et al. 3d hand pose detection in egocentric rgb-d images. In *Computer Vision-ECCV 2014 Workshops*, pp.356–371. Springer, 2014.

Hand Pose				
Intersecting(Y/N)	No	No	No	Yes
Intersecting Pairs	None	None	None	(9,12)
Penetration Depth(mm)	NA	NA	-1.21(less than threshold)	-8.04

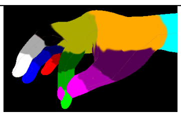
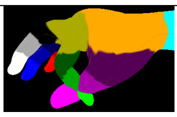
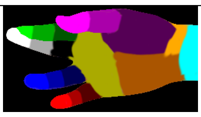
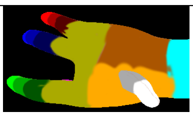
Hand Pose				
Intersecting(Y/N)	Yes	Yes	Yes	Yes
Intersecting Pairs	(9,12),(10,12)	(9,12),(9,13),(10,12),(10,13)	(6,9),(7,10),(8,11)	(6,17),(7,17),(8,17)
Penetration Depth(mm)	-10.69, -8.59	-4.41, -9.32, -5.73, -9.04	-11.62, -9.24, -6.52	-16.55, -5.24, -15.70

Table 3: Intersection detection and penetration depth computation for various hand configurations. The results show that our method detects self intersections in all possible hand pose configurations.

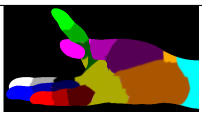
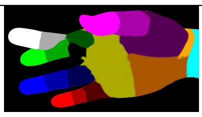
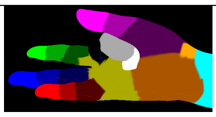
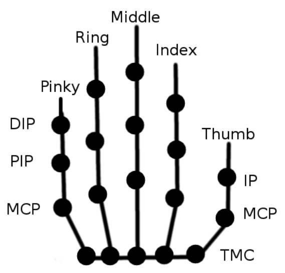
Hand Pose				Hand Parts
Joint Angle Limits	Index finger MCP: 90° Thumb TMC(FE): 50° TMC(AA): 45°	Middle Finger MCP(FE): $\pm 30^\circ$ Index finger MCP(FE):90° MCP(AA):$\pm 30^\circ$	Middle finger MCP: 90° DIP: 110° PIP: 90°	 Middle Ring Index Pinky DIP Thumb PIP IP MCP MCP TMC
Actual Joint Angles	Index finger MCP: 60° Thumb TMC(FE): 2.5° TMC(AA): 12°	Middle Finger MCP(FE): -15° Index finger MCP(FE):8° MCP(AA):15°	Middle finger MCP: 79° DIP: 110° PIP: 85°	
JAM	No Intersection	No Intersection	No Intersection	
Ours	Intersecting	Intersecting	Intersecting	

Table 4: Examples of poses where Joint angle constraints are satisfied but the poses are self intersecting. Here Joint Angle Method clearly fails. Our method detects intersections correctly. AA: Abduction-Adduction, FE: Flexion-Extension of joints, JAM: Joint Angle Method.

[12] H. Liang et al. Parsing the hand in depth images. *IEEE Transactions on Multimedia*, vol.16, no.5, pp.1241–1253, 2014.

[13] C. Choi et al. A collaborative filtering approach to real-time hand pose estimation. In *Proc. of the IEEE International Conference on Computer Vision*, pp.2336–2344, 2015.

[14] C. Xu and L. Cheng. Efficient hand pose estimation from a single depth image. In *IEEE International Conference on Computer Vision (ICCV)*, pp.3456–3462, 2013.

[15] M. Šaric. Libhand: A library for hand articulation. *Version 0.9*, 2011, http://www.libhand.org/

[16] M. Schröder et al. Real-time hand tracking using synergistic inverse kinematics. In *IEEE International Conference on Robotics and Automation*, pp.5447–5454, 2014.

[17] S. Sridhar et al. Interactive markerless articulated hand motion tracking using rgb and depth data. In *IEEE International Conference on Computer Vision (ICCV)* , pp.2456–2463, IEEE, 2013.

[18] J. S. Supancic III et al. Depth-based hand pose estimation: methods, data, and challenges. *arXiv preprint*, 2015.

[19] E. Tan. Estimating human tactile resolution limits for stimulator design. *Master's Report, Dept of EECS, UC Berkeley*, 1995.

[20] D. Tang et al. Real-time articulated hand pose estimation using semi-supervised transductive regression forests. In *IEEE International Conference on Computer Vision*, pp.3224–3231, 2013.

[21] J. Taylor et al. User-specific hand modeling from monocular depth sequences. In *IEEE Conference on Computer Vision and Pattern Recognition* , pp.644–651, 2014.

[22] J. Tompson et al. Real-time continuous pose recovery of human hands using convolutional networks. *ACM Transactions on Graphics (TOG)*, vol.33, no.5,pp.169, 2014.

[23] G. Van Den Bergen. Proximity queries and penetration depth computation on 3d game objects. In *Game developers conference*, vol.170, 2001.

[24] X. Zhou et al. Model-based deep hand pose estimation, In arXiv preprint arXiv:1606.06854, 2016.

[25] A. Tagliasacchi et al. Robust articulated-icp for real-time hand tracking. In *Computer Graphics Forum*, vol.34, no.5, pp.101–114, Wiley Online Library, 2015.