Procedural Hand Pose

By @jorgejgnz

Content

What is this?

ProceduralHandPose is a Unity plugin that helps you to pose hands easily for any object, helping you to complete modeling/animation tasks related with hands faster and improving your productivity.

Tested on Unity 2019.3.4f1.

Requirements

For an object to be compatible with ProceduralHandPose it has to be a Rigidbody with colliders. ProceduralHandPose will use collision detection to control when a bone is touching the object. The better the colliders fit the mesh the more accurate the pose will be.

Elements

There are 3 main elements involved in hands posing with ProceduralHandPose:

- Ghost hand: This is the hand that is posed. It will try to find a realistic pose according to master hand position/rotation, pose point position/rotation, rigidbody position/rotation/scale, pose point model and generator configuration.
- Master hand: This hand defines the initial state from which simulate the grabbing. You can rotate/move this hand as you wish and that will affect pose generation. Ghost hand will copy transforms from its master so moving master's bones independently can lead to longer/shorter bones in the ghost hand. Rotating its fingers will not have any effect on pose generation.
- PosePoint: This point defines the goal or destination that the ghost hand will try to reach.

Code structure

ProceduralHandPose comes from HPTK and its code structure. Each master-ghost pair and each pose point have their own data model (PosableHandModel.cs or PosePointModel.cs) and their own controller (PosableHandController.cs and PosePointController.cs). Master and ghost hands (and even their fingers and bones) have their own data model. This leads to easily understandable code and easier maintenance.

Controllers are inherited from their corresponding EventHandlers. From a Controller you can access directly to the Model but not from its EventHandler. EventHandlers are the components that must be shared between objects. EventHandlers enable other components to listen to events invoked by the Controller or to invoke EventHandler's events to change the Model. EventHandlers also include the ViewModel through which other scripts (that are not the Controller) must read the model.

PosableHandModel

- .posePoint: Destination that the hand will try to grab
- .configuration: Scriptable object that contains parameters to control pose generation. You can use the same configuration file for all (changing that file will affect all hands) or having a specific configuration object for every hand.
- .startPose: Initial pose. You may need to change this in very specific scenarios.
- .endPose: Destination pose. Set Fist pose to grab, set IndexPinch pose to pinch.

Poses are ScriptableObjects that you can find in ProceduralHandPose/Poses.

Configuration object

- Control
  - .maxDistance: Maximum allowed distance between master hand and pose point.
- Performance
  - .maxIterations: The more iterations the better results but it can affect performance.
  - .onlyFingerTips: Should collision detection consider the complete finger or just its fingertips?

PosePointModel

- .rotationMode:
  - *None:* Master hand rotation will determine ghost rotation.
  - *MatchXY:* Ghost hand palm rotation will match XY axis of pose point.
  - *MatchNormal:* Ghost hand palm normal will match Z axis of pose point.
- .behaviour:
  - *Default:* Ghost hand palm will try to reach the raycast hit point between master hand and pose point.
  - *ClosestPoint:* Ghost hand palm will try to reach the closest point of the closest collider found between master hand and pose point.
  - *AlwayMatch:* Ghost hand palm will match pose point, even when there are colliders between pose point and master hand.
- .minDistance: Distance between hand palm and found grabbing point
- .startAtLerp: Change this value for hook grabbing scenarios.

- .stopAtLerp: Change this value to make grabbing look stronger or weaker.
- .boneThickness: Change this value to control penetration in collision detection. You can set this value for each pose point.
- .collideWithOtherRbs: Should the hand collide with other rigidbodies apart from the rigidbody of the specified pose point?
- .collideWithTriggers: Should the hand collide with triggers?

How to change ghost destination (hand's pose point)?

Manually: Go to the PosableHandModel component for the hand you want to generate the grabbing. Change PosableHandModel.posePoint by any other PosePointHandler in the scene.

From your own script: To minimize code coupling, it is recommended to call handler's events to change the model (excepting from the controller that can access that model directly). This is an example:

```
public class YourScript: MonoBehaviour
{
	public PosableHandHandler hand;
	public PosePointHandler newPosePoint;

	private void ChangePosePoint()
	{
		hand.OnSetPosePoint.Invoke(newPosePoint);
	}
}
```

I have my own rigged hand, can I use this to pose my own rigged hands?

There is no code included to perform this task but it is easy to solve:

1. Make a copy of your rigged hand. One will act as master and the other will act as ghost.
2. Move every bone in master hand to match position/rotation of its corresponding bone in your rigged master hand. You can do this programatically by going through the vector HandModel.bones[i].transformRef.

   ⚠ This vector is only containing elements during PlayMode. Its content is filled when the game starts.
3. Let ProceduralHandPose to generate a pose.
4. Move every bone in your rigged ghost hand to match position/rotation of its corresponding bone in ghost hand. Again, you can do this programatically by going through the vector HandModel.bones[i].transformRef.

By doing this you can even pose hands with less than 5 fingers. If this feature is requested I can include a script to make this easier.