PrimeSense

Giving Devices a Prime Sense

PrimeSense™ NITE Algorithms 1.5

Disclaimer and Proprietary Information Notice

About This Guide

This guide's is intended for game/UI developers using the NITE middleware.

This guide contains the following chapters:

- **Chapter 1, Control by gestures,** on page 1-1, contains algorithmic notes regarding the mechanism for control by gestures.
- **Chapter 2, User segmentation,** on page 2-3, contains algorithmic notes regarding the user segmentation mechanism.
- **Chapter 3, Skeleton tracking,** on page 3-4, contains algorithmic notes regarding the skeleton tracking algorithm.

Table of Contents

Table of Figures

Support

PrimeSense has made a significant investment in the development of the user-friendly software and hardware development environment provided by the Prime Sensor™ Development Kit (PSDK). This development kit is accompanied by documentation that teaches and guides developers through the process of developing applications.

PrimeSense also provides an online technical support service for customers. This service provides useful tips, as well as quick and efficient assistance, in order to enable you to quickly resolve any development issues that may arise.

Getting Technical Support

PrimeSense's primary support email address is support@primesense.com. Each email that is received is automatically forwarded to a relevant support engineer and promptly answered.

Glossary

Term	Description
OpenNI	Standard Natural Interface Infrastructure.
Prime Sensor™	The brand name behind PrimeSense's products. It refers to the reference design for a 3D camera.
Prime Sensor™ IC	The chip developed by PrimeSense that is implemented in the 3D camera.
PSDK	Prime Sensor™ Development Kit.

This page was intentionally left blank.

1 Control by gestures

1.1 Basic assumptions

- The controlling hand is inside the field of view, and is not occluded by other users and objects. Exiting the field of view may cause losing the control.
- Working distance of the controlling user is 1m - 3.5m

1.2 Focus gesture

1.2.1 General

In order to start controlling the device, you must first gain control. Gaining control is done by performing a special gesture called 'focus gesture'. There are two supported focus gestures: 'click' and 'wave'. In the 'click' gesture, you should hold your hand up, push your hand towards the sensor, and then immediately pull your hand back towards you. In the 'wave' gesture, you should hold your hand up and move it several times from left to right and back.

Once you gain control, you are controlling the device, and until you release it, no one else can gain control. Release of control is application-dependant; for example, it can happen when the hand point is lost, or when it leaves a pre-defined working space.

- Try to keep the hand that performs the gesture at a distance from your body.
- Your palm should be open, fingers pointing up, and face the sensor.
- The movement should not be too slow or too fast.
- The 'click' movement should be long enough (at least 20cm).
- Make sure the 'click' gesture is performed towards the sensor.
- The 'wave' movement should consist of at least 5 horizontal movements (left-right or right-left)
- If you have difficulty to gain focus, try to stand closer to the sensor (around 2m), and make sure that your hand is inside the field of view.

1.2.2 Focus gesture output

When focus gestures data is reported to the application, it contains the status (gesture started, gesture completed), the gesture type ('click, 'wave'), and the location where the gesture occurred.

Supported gesture types:

- Click
- Wave
- Swipe left

- Swipe right
- Raise hand candidate
- Hand candidate moved

Supported status:

- **Gesture started** – Occurs when a focus gesture is started. For the ‘click’ gesture, this happens after the hand did a large enough movement forward. For the ‘wave’ gesture, this happens after the hand did 2 waves.
- **Gesture completed** – Occurs once the entire gesture is recognized by the framework.

Only the ‘click’ and ‘wave’ are meant to serve as focus gestures.

The ‘swipe left’ and ‘swipe right’ gestures are for allowing immediate limited control of the system without performing a focus gesture, but they have a high false rate and should not be used as focus gestures.

The ‘raise hand candidate’ gesture and the ‘hand candidate moved’ gesture are also not meant to serve as focus gestures. The ‘raise hand candidate’ gesture occurs each time a user raises his hand.

The ‘hand candidate moved’ gesture occurs each time a user moves his hand.

The last two gestures can be used to help the application provide relevant feedback to the user.

1.3 Hand tracking

The controlling hand is the hand that performed the focus gesture.

The output is the hand’s position in real-world coordinates in units of mm in each frame.

- Try to keep your controlling hand away from other objects, including your other hand, your head or your body.
- Avoid touching other objects with your controlling hand while you are in control.
- If another object is close to the hand, the hand point might move to the other object.
- It may happen that a hand point is lost.

2 User segmentation

2.1 General

The purpose of user segmentation is to identify and track users in the scene. Each user in the scene is given a unique and persistent ID, and the main output of the user segmentation process is a label map giving user ID for each pixel.

The skeleton tracking algorithm uses this label map to generate a skeleton. Any inaccuracies produced by the user segmentation algorithm will manifest themselves as possible inaccuracies in the pose tracking.

2.2 Known issues

- Some difficult scenarios that may produce inaccuracies in the user segmentation include:
    - User moving while very close to objects in the scene such as walls, chairs, etc.
    - Two users moving while touching,
    - Moving the sensor while user segmentation is active
- Under occlusions or situations where the user exits the field of view, their ID may be lost or incorrectly swapped with another user's.
- If the user is not visible for more than 10 seconds, the user is considered lost and an event is generated by the API. Their ID is freed and they will be given a new ID if they are re-detected. Since IDs are recycled, another user may get the lost user's ID at some point in the future. It is therefore important to catch this event and inform the pose tracker to stop tracking the old user. This will ensure that when a user with that same ID is re-detected, the pose tracker will request re-calibration on that user.

3 Skeleton tracking

3.1 Basic assumptions

- User's upper body is mostly inside the field of view. Try to avoid occlusion by other users and objects.
- Ideal distance is around 2.5 m.
- For better results, user should not wear very loose clothing. Long hair may also degrade the quality of the tracking.

3.2 Automatic Calibration

Auto-calibration enables NITE to start tracking a user, without requiring a calibration pose. In NITE versions before 1.5, the user had to stand in a calibration ("psi") pose before the system could build his skeleton. Now the skeleton can be created shortly after the user enters the scene.

The quality of the skeleton improves with time. Although the skeleton of the user appears almost immediately, the auto-calibration process might take several seconds to settle at accurate measurements. As a result, the skeleton might be noisy and less accurate initially. After auto-calibration determines stable measurements, the skeleton output becomes smooth and accurate.

3.2.1 Usage

Auto calibration is employed by default. In order to cancel it and use the "psi" calibration, set "UseAutoCalibration=0" in FeatureExtraction.ini under [LBS].

3.2.2 Limitations

- Auto calibration works on standing users. A sitting user will not be calibrated automatically.
- Most of the user's body should be visible in order for the automatic calibration to take place.
- The user should be located further than ~1m from the sensor in order for the automatic calibration to start.

In the current implementation, after stable measurements have been computed (a process which may take a few seconds), they remain fixed for the remainder of the skeleton's lifetime.

3.2.3 Recommendations

The following recommendations will help to complete the calibration faster and achieve better results. The recommendations apply to the auto-calibration phase only (first seconds of the user in the scene) and not to the subsequent frames.

- The user should be facing the sensor and standing upright.
- The hands should not hide the torso area.
- The user should avoid fast motion.

3.2.4 Load/save calibration

Since previous versions of NITE did not support automatic calibration, NITE provided the ability to save and load the calibration data. Automatic calibration makes this feature obsolete. However, the load/save calibration is still supported for backwards compatibility.

If automatic calibration is active (the default), both save and load will fail.

3.3 Skeleton output

The API returns positions and orientations of the skeleton joints. Note that the lengths of the body segments (e.g. distance between returned elbow and shoulder) might vary during tracking.

3.3.1 Differences between joint positions and orientations

The joint positions are, in a way, more accurate than the joint angles. To explain this point further we will compare two ways of computing a hand position in your game. One way is to take the position of the hand joint from our API. The other is to take the torso position & orientation, together with shoulder and elbow joint angles, and use all of these to drive an avatar rig. Typically the resulting hand position will be *different* from the one returned by our API because your avatar will have different segment lengths than the model used in our skeleton API (especially considering the previously mentioned point that our skeleton allows body segment lengths to vary over time whereas avatar models in games typically have fixed lengths). The positions would match if the segment lengths matched exactly all of the time.

Beyond just being *different,* the hand position computed using the latter angle-driven approach will typically be more *noisy* than the hand position returned directly from our API. As a result we recommend using the joint positions whenever possible. Unfortunately, this is usually not very practical for avatar-based games that need to be driven using joint angles. In such cases the available options are to either use the joint angles and deal with the noisier hands and feet, or to use the joint position as (soft) constraints in a post-processing IK/retargeting step which will compute modified joint angles better tuned to your avatar..

3.3.2 Additional notes

- A confidence value is returned for the individual joints but it is fairly basic at the moment – it is 1 while tracking seems to work, 0 if tracking fails and we're uncertain about the output. If skeleton heuristics is enabled (see Section 3.4), joints will receive a confidence of 0.5 when the heuristics adjusted their position or orientation.
- Note that if the confidence is 0 there is no guarantee the corresponding position/orientation is meaningful. Sometimes it will be kept at its old value, but other times it might be meaningless.
- In cases where a joint's orientation is not known, the feature extractor tries to give its best guess. For example, when an arm or leg is straight, its full orientation is unknown as it is not really possible to detect its "twist" using the depth sensor. Also the lower arm and lower leg's "twist" cannot be reliably detected in most cases.
- The orientations of the hand and foot joints are not currently independently tracked. They are simply set to be equal to the respective elbow and knee joint orientations.

3.3.3 Joint transformations defined

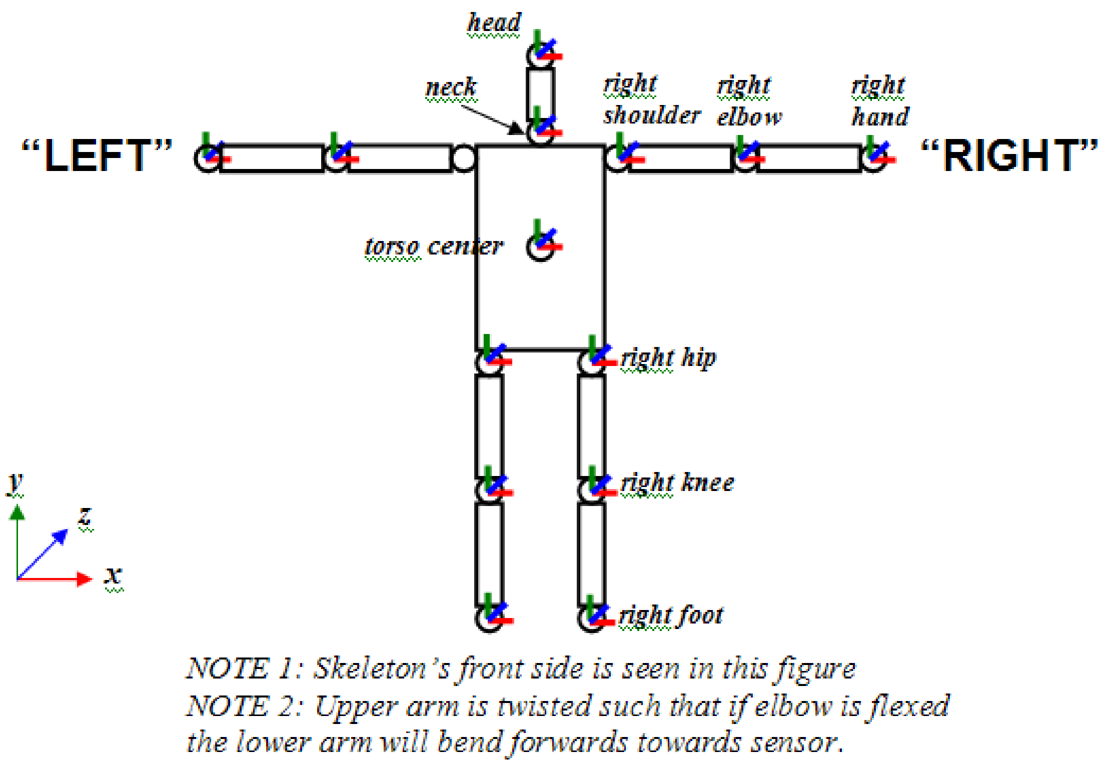
head
neck
right shoulder
right elbow
right hand
"LEFT"
"RIGHT"
torso center
right hip
right knee
right foot
y
z
x
NOTE 1: Skeleton's front side is seen in this figure
NOTE 2: Upper arm is twisted such that if elbow is flexed the lower arm will bend forwards towards sensor.

Figure 1: Joint definitions

The figure above illustrates the coordinate system and skeleton representation for a user facing the sensor. These will be explained in more detail below. But first we mention an important note about our definition of “LEFT” and “RIGHT” sides of the skeleton.

Suppose you stand in front of the sensor, facing it in a T pose as shown above. If you look at the raw depth map output from the sensor, *assumed to be in **Mirror mode,*** you will see a mirror-like reflection of yourself on screen – move your right arm and in the depth map the arm on the right side of the depth map moves as well. The skeleton algorithm calls this the “RIGHT” side. This gets confusing if you think of it from the perspective of the rendered skeleton (avatar). If you imagine the figure above is a rendered 3D avatar facing us in a T-pose, the side we labeled “RIGHT” is actually the left side of that avatar. Nevertheless, we define this side to be “RIGHT”, and the opposite to be “LEFT”.

Joint positions and orientations are given in the real world coordinate system. The origin of the system is at the sensor. +X points to the right of the, +Y points up, and +Z points in the direction of increasing depth. The coordinate frame is shown in the figure above.

Joint positions are measured in units of mm.

Joint orientations are given as a 3x3 rotation (orthonormal) matrix. This represents a rotation between the joint’s local coordinates and the world coordinates. The first column is the direction of the joint’s +X axis given as a 3-vector in the world coordinate system. The second column is the +Y axis direction, and the third column is the +Z axis direction. Our “neutral pose” is the T-pose shown in the figure above. In this pose, each joint’s orientation is aligned with the world coordinate system. That is, its orientation is the identity matrix.

3.4 Skeleton Heuristics

“Skeleton heuristics” refers to a set of heuristics that provide a more plausible skeleton behavior when joints have zero confidence. Joints may have zero confidence in situations where they are lost or occluded. Since joints with zero confidence have undefined positions/orientations, the heuristics are meant to fill in these undefined values with reasonable values. These heuristics include:

- If an arm has zero confidence, it is adjusted to point downwards at the side of the body. Its joints are set to have confidence 0.5.
- If a leg has zero confidence (especially common for the far leg when the skeleton is side facing), it is also adjusted to be downward pointing with confidence 0.5.

- Whenever a joint transitions between tracking and heuristics, its transition occurs in a smooth fashion over a number of frames, to avoid rapid jumps.

  One of the results of running these heuristics is that when a limb is lost or occluded, you will see it transition into its heuristic configuration. This may appear strange in some cases, where the heuristic configuration is far from where it should be, but this is a normal side-effect of applying these heuristics.

  Currently the heuristics are only applied if the *whole* limb has zero confidence. If only the lower limb has zero confidence, it is not adjusted with any heuristic, and remains with its original low confidence.

  Usage:

  Since NITE 1.4, skeleton heuristics is enabled by default. It can be explicitly enabled by
  XnStatus status=userNode.SetIntProperty,"SkeletonHeuristics",255);

  And disabled by:

  XnStatus status=userNode.SetIntProperty,"SkeletonHeuristics",0);.

  Disabling skeleton heuristics allows you to perform your own handling of joints with zero confidence.

3.5 Known issues

- Arm tracking is less stable when the arm is close to other body parts, especially the torso. If both arms are close to the torso, as well as to each other, they might get mixed up.
- Leg tracking is still somewhat unstable and noisy. It works better when the user stands with legs separated. Fast motions and complex kicks or crouches might cause the tracking to fail.
- Pose tracking may also become somewhat unstable if the head is not visible.
- Arms and legs in extremely stretched positions (i.e. near the limits of human flexibility) might be lost by the tracker.
- If the skeleton is stuck in a faulty pose, or stuck facing the opposite direction, then returning to a "simple" pose (arms away from torso so the sensor can see them and legs separated) should help resolve it.
- In general, very fast motions may cause tracking failure.
- In some cases, overall tracking might be bad. Re-calibrating the user may resolve the problem.

This page was intentionally left blank.