Salient Features of the Project

This project allows the user to ‘control’ a character in a 3D virtual world by giving him certain ways to control the character’s position and orientation and also to interact with objects in the scene.

The character has the following abilities -

1) Swinging the hands of the user at a certain threshold frequency triggers movement of the character. The rate of hand swing determined from the live user feed determines the type of animation to be played - walk/run. This animation drives only the lower part of the character while the upper part of the body is taken directly from the live feed.

2) The user can raise either of his legs to turn in the opposite direction i.e. on raising the left ankle to the side the user turn a certain amount to the right - the amount being decided by the magnitude of the raise. Turns can be taken during idle/walk/run.

3) The user has the ability to grab items as well. This is a difficult task to do in a system like this. From the users perspective it is not easy to get an idea about the position of objects in the virtual world, and make a good judgment of where to put one’s hand to reach that object. As such a simple implementation of collision detection, although sufficient, may not be effective enough and neither will it look convincing. The forward kinematics system does its best to reach out to the desired object i.e. even if the object might actually be out of range the limbs make their best effort (get fully stretched in the direction of the target object). This affects the root/shoulder/elbow orientations to allow the hand to “touch” the object.

For our system - the user has the ability of ‘lighting’ candles using his fire-powered hands.

4) The character collides with other rigid objects such as boxes if they come in the way, triggering the NVidia PhysX engine.

5) Another gameplay element involves collecting spheres by “touching” them with your hands, which increases the players score.

6) Inspired by Mario, the player can also jump and bump his head onto suspended blocks (but as it turned out, for the demo we had put the blocks a bit too high and could not show the effect).
A brief overview of the underlying system:

- The lowest layer is the PrimeSensor device, which receives raw sensory data – a stream of depth images from the Kinect.
- The next layer, which is C-shaped, represents OpenNI. OpenNI is the communication interface that interacts with both the sensor’s driver and the middleware components, which analyze the data from the sensor.
- The NITE Algorithms layer is the computer vision middleware. It processes the depth images produced by the PrimeSensor and generates a skeleton and populates the various joints with position/orientation values.
- A Unity Wrapper that is used for the Unity3d – NITE/OpenNI integration.

Fig 1. Layer View of the system

Fig 2. Skeleton derived from NITE
Details on Gesture recognition

1) Walking:
We note the position of the hand (in the character this corresponds to a point close to its wrists) and we calculate the frequency of swings by the number of times the hand crosses over from the back to the front of the root and vice versa. We use a weighted average over the past few frames and decide the current rate. When this rate is beyond a certain threshold we start the character animation. This rate determines whether the character uses walk or run animation.

2) Turning:
Since the hands are already engaged in walking we decided to make use of the legs for this action. We take position of the ankle (projected on the torso-body plane) and depending on how fast this changes we swing the character. It is important to note that only an outward swing (the legs are taken away from the center of the body) produces the turning motion.

3) Jump:
We note changes in the torso joint position in the skeleton that we get from NITE and make appropriate changes to the game character’s torso position.

4) Pyrokinesis:
(The gesture was implemented but we ran out of time while generating a suitable ‘action’). Our idea was to generate an action similar to this YouTube video. We track the shoulder-elbow and elbow-hand bones and when they are nearly orthogonal to each other with the hand pointing forward this action gets activated.

Choices we made:

1) Not to learn Hidden Markov Models - Gesture recognition without using probabilistic models cannot scale. Since our focus was limited to a few gestures and the existing complexity of setting up and working with a whole stack of libraries, we decided to implement all the gesture recognition in the form of hacks. In hindsight, we think it was a sensible decision considering the time limitations.

2) Using a custom character: We could have gone with a bunch of sticks and spheres for the character, but decided to use one with a bunch of simple animations attached to it. It helped us get something up and running really quick, but hit us hard in the variety of things the character could do. Difficulties in retargeting other animations such as climbing ropes and swinging limited the capabilities of the soldier character - this would have required us to build our own character from scratch and rig it first.
**Choices we were forced to make:**
There were several difficulties we faced in the implementation process.

1) Sampling:
By far the trickiest of them all, and the one we underestimated the most was sampling. It came up over and over again and we should have made better design decisions before hacking up the code.

2) Blending:
The initial idea was to blend the existing animations with the live feed from the Kinect. Unfortunately, our limited expertise with Unity3d was exposed, and all we were able to accomplish was a 0-1 blend. When the walking motion is triggered, the top half of the body (head, hands and root) utilizes the live feed and the bottom half (hips and legs) uses the walk animation. Ideally, we’d have liked to use different weights for different joints in the blend process. While blending in Unity3d is trivial when existing animation files are the input, it isn’t clear how to go about doing this with a bunch of animation files and the live feed simultaneously.

3) Forward Kinematics with "Matrix style" effects:
The idea was to slow down time and make the camera focus on that part of the body undergoing FK. This required us to solve the FK problem on temporary copies of the ‘transform' object (this stores everything from local/global position/orientation, matrices etc.) of the joints involved, and then SLERP’ing them over at a later point. However, Unity at presents supports ONLY copy by reference of its objects. We tried deep copying the objects but most of the class variables turned out to be read-only. Therefore this idea failed to work.

4) Unsynchronized walking:
Synchronizing the legs and hands during walking makes the motion look a lot more authentic. In theory, it seems very simple: figure out when the player’s left hand is at the peak of the swing, and make sure the right leg lands at that time. It was more of an implementation hurdle, since it isn't apparent even now how one accesses the animation key-frame data in Unity3D. Hardcoding frames and their significance led to jerky motion, so we decided it was better off if we just played the walk/run animation from the 0 frame every time the user triggered the walk gesture by swinging his hands.

**Performance of the gesture recognition**

The walking motion is pretty decent - it doesn’t require too many swings to get off the block and neither do you need to swing too fast to walk. While walking the character is able to interact with objects in the scene, such as the candles, boxes and spheres, but cannot turn. Since the camera is attached to the root of the game character, and it turns when the root turns even a little bit, the walk motion may falsely seem to sway from side to side sometimes.
The turning gesture we came up with wasn’t the best one, but it certainly added an element of fun to the game (We initially had the idea of turning when the user bends – but performing such a pose while swinging arms is pretty difficult). Having multiple walking and turning gestures seems like a good idea, since different interactions would be possible. For example, if lifting legs in a wave like fashion represented walking, then the hands would be free to interact with the environment. The hands could also be used for turning in this case.

The FK does work out nicely. Lighting the candles is pretty simple. One doesn’t have to do the difficult task of estimating where exactly he has to move his hands in the user space to light the candle in the game world. If we had got the slow-mo ‘Matrix’ effect to work then this part would have looked much more interesting and convincing.

**Challenges of using Kinect**

1) Data, data and lots of it
Kinect generates a bunch of depth images every second and these populate a base skeleton (as described before) that we use as the skeleton for our character. The problem that we face is deciding what rate to sample the skeleton updates for gesture recognition. Sampling is gesture specific and the rate of sampling greatly determines the effectiveness of the recognition system. For example, one of the problems we face is during identifying hand swings for the walk motion. We find how often each hand crosses over the root-body plane. If the sampling rate is kept very high - the walking is a bit jerky, with the character starting and stopping numerous times. If the sampling is slower the character mostly overshoots its stopping point. (In both cases we use some kind of weighted joint information from the past few frames).

2) Limited skeleton structure
The NITE API returns positions and orientations of only a few skeleton joints. For example, in the skeleton hierarchy the root is directly connected to shoulders and the neck for the upper torso. This prevents accurate representation of intermediate points such as spine bones. Again the current implementation of NITE API does not provide joint information for wrists, fingers and toes. The amount of data that the Kinect sensor can send is limited by the bandwidth of the USB port, and the 640x480 resolution of the depth image makes accurate finger/toe tracking difficult. As a result it is impossible to get full control of the body at any point of time and or generate complicated gestures, say involving fingers. Alternative approaches such as painted gloves or glow points at the finger tips can be used if finger tracking is absolutely necessary.

3) Similar poses - different gestures
An example will clarify the idea. Consider the pose in which the user has his hands in the ‘rest’ position i.e. his hands are hanging vertically down and resting along his palm resting on the sides of thigh. Now this pose can at least come up come during two cases (considering gestures we studied) - the “idle position” and the “walking gesture” when the arms swing past the root-body plane. If we don’t consider the actual velocity of the hand swing, the algorithm will
identify both the cases to be idle poses - and one can observe the game character suddenly stopping in between walking.

4) Complexity of Gestures
Identifying gestures using a bunch of dot/cross products and some numerical methods is not perfect and robust. Too many parameters need to be specified to recognize each particular gesture. To make the system better the need a machine learning approach.

“Why an understanding of human motion and animation helps you do a better job with working with it.”
We are so used to making gestures in our day to day life that we never give them a second thought – there are a lot of unique combinations of joint positions and orientations that make them possible. Segmenting each of the gestures to simple instances of joint translations and rotations make gesture recognition possible. Therefore understanding human motion is very important for success with gestures.

Knowledge of animation techniques allowed us to do blending, solve forward kinematics problem and leverage the strength of quaternions over Euler angles and use concepts such as slerps for interpolation.

Reading
We did not borrow any ideas from a particular paper as such. A lot of reading involved understanding the NITE documents. Some other papers we read were -

1) Randomized Trees for Human Pose Detection

2) Microsoft's algorithm used in the Kinect:

3) A Bayesian Framework for Human Body Pose Tracking from Depth Image Sequences
http://www.mdpi.com/1424-8220/10/5/5280/pdf

4) 3-D human body tracking from depth images using analysis by synthesis
http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=958455

5) Object Class Segmentation using Random Forests
http://www.robots.ox.ac.uk/~vgg/publications/papers/schroff08.pdf

6) A Gesture Recognition System Using 3D Data