Getting RealSense 3D Characters into Your Game Engine

1. Introduction

Some of the best advances in technology happen when we close the gap between human and machine, creating bridges which allow us to communicate with computers in our own language. In recent years this has manifested in the form of touch screens, gesture and voice recognition, face tracking and fingerprint scanning, all of which take what we do in the real world and convert it into something the computer can process. The possibilities are exciting for game development.

![Figure 1: Use the Intel® RealSense™ SDK and scan yourself directly into a games engine](image)

One of the next big leaps in this trend will be full world scanning, through **perceptual computing**, giving the computer a complete 3D map of its environment and everyone in it. As you might imagine, the possibilities are endless when the computer knows as much about what is going on around you as you do. In fact, it’s highly likely the computer will be simultaneously aware of everything around you, compared to the human user who has to focus on a subset of their world.

We are not quite at the stage described above, but we’re certainly on the path towards it and the journey will bring incredible opportunities for pioneers to create that future. In the Intel® RealSense™ SDK, you will find a number of samples which contribute directly to this work, such as the 3D object scanner and the segment detector. Scanning and then processing this new type of data is the starting point for giving the computer a true sense of its place in the world.
This article will add to the work mentioned above by describing how you can use your forward facing depth camera to scan someone and then generate an entire 3D character based on the data obtained. We will then place this character into a virtual 3D world and controlled by the computer.

As a prerequisite, you should have some familiarity with basic conversion of depth data to 3D point cloud data which you can obtain from my previous article entitled “Intel® RealSense™ Technology and the Point Cloud”, and also some knowledge of 3D graphics principals such as vertices, texture mapping and elements that make up a typical 3D game or simulation.

2. Why Is This Important

In order for the computer to make decisions about the world around it, it needs to know what that world looks like and a 2D image will not be sufficient in many cases. The computer needs to look around and have a sense of scale and context, and that can be best achieved by creating a 3D representation of that world.

The techniques explored here will be the same techniques used to create much of this 3D world and give you a head start in solving the next set of problems when new hardware appears. We will shortly be introduced to real-time rear facing depth cameras, and who knows what kind of technology we’ll get after that? Rest assured the future will bring some exciting advances in this field and it all begins with scanning in a single person and stitching them into a small 3D world.

From a more pragmatic point of view, if you are investigating this technology to allow scanning of 3D objects and getting them into your own software, this article is a good place to start. You will learn the scope of the Intel RealSense SDK, and what additional coding is required to achieve your own goals. You will also learn about 3D geometry stitching, which can be applied to solve problems not specifically tackled here.

Not everyone will be writing their own games engine, but understanding the process will allow you to see past any shortcomings you may face out there in the real world of games development, and give you an edge when it comes to solving the trickier challenges you might face.

3. Making Your 3D Character Model
You will appreciate that a person sitting at their desktop computer is not going to present the depth camera with a full skeletal view of the person to be ‘virtualised’, and at most the head, shoulders, upper body, arms and hands will be visible. It makes sense then that the remaining parts of the 3D character need to come from more traditional sources. In our case we will be using a standard civilian biped character model with a detachable head.

![3D Character with Separated Head Mesh](image)

**Figure 3: This 3D character is modelled with a separate head mesh**

The benefit of this approach is that we can throw away the head provided by the artist and replace it with a head generated from the 3D data coming from the depth camera. In my [Point Cloud article](#), I have extensively covered how this can be done from scratch, using nothing but the raw depth data to generate an accurate real-time depiction of what lies in front of the computer. In this case, such a result would have artifacts we do not want such as indeterminate facial edges, bad hair and no rear section. These are not qualities we want in a 3D representation of our virtual selves, so we must use an alternative approach which eliminates these errors.

The solution is to start with an existing head mesh with predictable areas of vertex data, and then modify that vertex data to reflect the contours of the 3D data coming from the depth camera. At the same time, the RGB color data from the camera is also converted to a texture image and applied to the head mesh with a combination of UV mapping and texture blending. For best results, a still capture is much better than a live stream as it will allow small changes and shifts to be made in order to create a good fit between the stock head mesh and the highly erratic 3D data you might have to deal with. The output will be a new head mesh and a new texture image that can be swapped with the existing 3D characters own head to produce a hybrid character.
Figure 4: Character textures would reserve an area for the face, eyes and ears

Of course this is just the first step towards creating a character that matches what you might look like in a virtual world, and the depth camera can once again help with this. Using the RGB color data from the camera you can establish a measurement of skin pigment and also the color of the shirt you are wearing. This simple RGB color sample is then used to make modifications to the rest of the character body, specifically arms and hands color and upper clothing color. As the computer has no way to scan your lower half, the color and style of pants, shoes and other adornments can be made manually. This manual configuration of characters is a pretty common feature of most RPG and many third person games, and is beyond the scope of this article.

Within the scope however is the automated recognition of skin color and shirt color, which can be done in a variety of ways, the easiest of which is to locate an ideal landmark from which to grab the color of the pixel at that position. For example calculating a centre point between the left ear and the left eye will locate a pixel of skin from the face, and taking a sample a few centimetres below the location of the chin that does not co-relate to the previously grabbed skin color will give you a color for the clothing. We can assume for the purpose of expediency that the color of the face is also the color of the arms, hands and other exposed parts of the 3D character.

Once you have your modified head mesh, head texture and a modified body texture, you are ready to assemble them all within the virtual world. In our case we have chosen a 3D character model that has been pre-built with a set of common animations for standing idle, walking, running and other actions you might associate with third person controls. From here, you can decide whether the character would be an avatar representing the protagonist of the game or one of the in-game characters who will meet the player during the game story.
4. Stitching Real-time 3D to an Existing Head Mesh

Now we have covered the general overview of the technique, we must look at the specific technical process of somehow grafting your actual face onto a model head and this is where some basic knowledge of 3D programming will be useful. Based on the prerequisite of having become familiar with the Point Cloud article, we can dispense with a break-down of obtaining the real-time 3D geometry and RGB texture color data from the depth camera and focus on how this data can be merged with a fixed size head mesh.

![Figure 5: A wireframe representation of a typical head modelled by an artist](image)

When you ask an artist to create a 3D character, they go to great pains to make sure the overall head and body do not take up too many vertices, as this improves overall game performance by providing fewer polygons to render. This leaves us with a head mesh that does not have the fidelity of the 3D data we can obtain from the depth camera, which can scan in 3D data with a resolution of hundreds of vertices in width. Modern games engines might provide a head with less than 50 vertices across, which means we will be missing a lot of the smaller details that make up your unique facial features.

Fortunately games development has a solution for this called Normal or Bump Mapping, which can take a high density grid of 3D points and convert them to a texture image that stores a vector that describes the direction of the surface being represented. More information on normal mapping, complete with illustrations, can be found from the following link at http://www.mat.ucsb.edu/594cm/2010/adams_rp1/index.html. Think of it like a cloth over a face and as the cloth contours over the facial features, the angle of the surface relative to the direction of the head is stored in a texture pixel. In games it is often more desirable to store high resolution data in textures rather than geometry as the hardware is optimized to render the results much more quickly. Thanks to this technique, even though the geometry of our head mesh might be low, we can retain all of the higher resolution 3D data from the real-time depth data and encode it in a normal map that will be associated with the texture of the head.
Texturing the head is much simpler, only requiring the RGB color data from the camera to be mapped to the correct location on the head mesh and then blended with the existing head texture, but it has a few complications. The texture for the head mesh is typically created by an artist, who will fill in details a single snapshot cannot capture, such as the side of the head or the rear. One technique is to blend the RGB texture from the camera into the head texture and feather the edges where the head mesh starts to wrap to the sides, top and rear. The downside to this technique is that you will have your face mapped to the front of the head, but the hairstyle, hair color, ears, side locks and back of head are entirely created by the artist and the result will range from mildly amusing to utterly inappropriate.

Figure 6: The result of using a single frontal shot to texture your head

A better way is to make several scans of your head, much like a mugshot but with more photos, and precisely controlled to allow the software to stitch it all back together again. The Intel RealSense SDK has provided a sample that does precisely this, and can be run immediately from the pre-compiled binaries folder.

Figure 7: The 3D object scanner example, after scanning a programmers head
Once you have this data, and more importantly, have the texture pre-mapped to the mesh produced, your 3D character will have everything it needs to represent the head from any angle, not just a forward facing pose. Naturally you will want to create your own 3D scanner to suit your project, so what follows is a break-down of how the technique could work for you.

Figure 8: Custom 3D head scanner created to demonstrate face stitching

There are a number of ways you could build a 3D head that looks correct from multiple angles. One is to start with a perfect sphere and just like a piece of clay mould it into shape by working out the orientation of the head and then scanning the forward facing depths returned from the camera and carving the detected depth into the sphere. With enough passes, the face will start to take shape and as the object sphere is a sealed object, when you have finished the head is a complete 3D object ready for your game or application. The downside to this technique is that head orientation tracking, such as the face normal data found in the SDK, is not perfect and can jitter quite a bit, meaning fine details like eyes, noses and mouth get blurred with repeated scanning, and the scanning itself can take a bit of time depending on the method you use.

A better technique, hinted at above, is to take several mugshots of the head, at different angles, and then work out how to weld them together to form a single usable object mesh. You could take the mesh shots separately and then attempt to manually connect them in a make-shift art tool or small program but given the tendency for the head to shift position and distance from the camera during the scanning session, this will only lead to frustration. If your scans are accurate you could blend the meshes against the first mesh you scanned and make adjustments if a vertex exists in the world space model data, and create a new world space vertex point if none previously existed. This will allow your head mesh to get more refined the more samples you provide, and potentially allow the whole head to be mapped, if you can find a way for the user to rotate 360 degrees while sitting at a desk. The downside to this technique is that getting your head scan to remain in a central position during scanning, and converting that 3D data to world coordinates creates some challenges.
Figure 9: Get head world positions wrong and you end up with two noses

The problem with live 3D scanning is that there is a human at the other end of it, which means they fidget, shift in their seats, lean back and move forward in subtle ways, and that’s not accounting for the subtle pitch, yaw and roll of the head itself.

The perfect technique is to detect signature markers within the scanned mesh, in order to get a ‘vertex fix’. Think of this as charting your position on the ocean by looking at the stars, and using constellations you can work out both your relative orientation and position. This is particularly useful when you produce a second mesh, which you can apply the same marker detection algorithm, find the same pattern and return the relative orientation, position and scale shift from the first mesh. Once you have this offset, adding the second mesh to the first is a simple world transform calculation and then adding the extra vertex data to the original mesh. This process is repeated until there are no more meshes you wish to submit, and then you proceed to the final step.

If the technique you chose involved layering vertex data onto a single object you will likely have a lot of surplus and overlapping vertices. These need to be removed and a new triangle list produced to seal the object and export in the format of your choice.

5. Exporting your 3D head and Loading into Your Game

It could be argued the hard work has been done, and the incredibly difficult job of converting a real head into a virtual one is over. The last big hurdle is to decide on a 3D file format and create a file that represents the data you have painstakingly created. Fortunately you do not need a format that supports animation, bone structures or even UV coordinates if you are storing your color data in the vertex format itself. The most popular 3D file format that fits this description is the Wavefront .OBJ file format which is a text based open format that produces a human readable sequence of data to describe a geometric form.

You will want to keep your exporter as simple as possible, handing over more complex operations to tools that can specialise in the extra elements you may want to introduce, such as bones for facial expressions. Perhaps in the future your 3D head scanner could detect actual mouth and eye
movement and determine necessary bone locations, providing the animation data so your characters can be pre-programmed with expressions created directly using the depth camera. In fact as little as 10 phoneme expressions are required to produce a convincing mouth movement for talking characters. Precisely how this can be done is beyond the scope of this article, yet the groundwork achieved from this article will prepare you for taking this pioneering step if you choose.

Saving your 3D model file simply requires creating a file, and writing out a few string lines encoding the vertex and triangle data. Here is a simple extract of the OBJ file format:

```
# OBJ Model File comment
# Object
o mesh1
# Mesh
g mesh

# Vertex list
v 3999.993408 999.998352 -0.000000
v 3999.993408 999.998352 -0.000000
v 3999.993408 899.998352 -0.000000
vt 0.000000 -0.000000 0.000000 -0.000000
vt 1.000000 -0.000000 0.000000 -0.000000
vt 1.000000 -1.000000 0.000000 -0.000000
vn 0.000000 0.000000 1.000000
vn 0.000000 0.000000 1.000000
vn 0.000000 0.000000 1.000000

# Face list
f 1/1/1 3/3/3 2/2/2
f 4/4/4 6/6/6 5/5/5
f 7/7/7 9/9/9 8/8/8
```

Most development tools support the import of OBJ 3D files and almost all 3D model viewers as well, so you can instantly test to see if your creation exported correctly. If you wanted to additionally export extra vertex data such as the diffuse component you will need to either create your own 3D file format for export and import, or implement export support for a more sophisticated format. One such format is the DirectX file format which also allows text exporting, but supports a much wider range of export attributes, including custom vertex data formats.

It is during the loading of your 3D head model that you will be required to associate it with the headless body you will already have loaded into your game engine. You or your artist would have created an anchor point or hot spot within the body model which locates the neck, and it is this location which fixes the head in place. Of course once the head is attached, you will suddenly realise the journey is far from over.
Figure 10: A first attempt at importing our 3D head onto a real body

As you can see in Figure 10, we have our head attached to the body, but it’s immediately apparent that because our scan only focused on the front and sides, there is precious little detail at the back and the top, which makes our character far from convincing.

6. Adding a Wig

Until we have a universal way for depth cameras to see around corners, or a method to spin the user in their seats, we will not be able to grab the back of a person’s head with a single camera. We can imagine wonderful new systems that could perhaps use a tablet to inspect all sides of your head and transmit that extra data to the scanner software, but for the purposes of this article we are confined to a single forward facing camera and a non-rotating user.

The solution comes from one of the many pages of the game developer’s bible, and one we can convince ourselves is a rather nice feature as opposed to a rather sinister hack. Having identified the top and back of the head contains dodgy geometry, we proceed to cover it up with something suitable. If your game was set in outer space, it could be a space helmet, or if you want to offer your end users the ability to customise the character you could provide a range of hair pieces which conveniently drape down the side of the head to hide the ears that don’t exist.
Figure 11: Suddenly the character belongs in the scene, and the 3D head looks great

It might seem like cheating, but if you are lucky enough to have an artist, they will probably be insisting that in order to achieve a good visual result, your 3D head is incorporated sensitively into the body of the character. When you start to study the options, you will find there is almost no scenario where your game characters cannot be wearing something up top.

7. Tricks and Tips

Do’s

- Storing the color data of your face scan in the diffuse component of the vertex data is a handy way to avoid the need for a texture, and offers other advantages when it comes to optimizing the final mesh
- If you really want to avoid adding hats and wigs to your final game character, tie or gel back your hair before making your scans. The flatter it is, the better the scan will be. Loose hair interferes with the depth camera and produces scatter reflections which do not scan well

Don’ts

- Do not proceed to implement the techniques in this article until you have a good practical grasp of 3D geometry coding, as manipulating variable oriented vertex data can be tricky
- Don’t forget to process the raw depth data to remove interference such as stray depth values that are not consistent with the surrounding depth pixel data. Fail to do this and all the work to create a good 3D head mesh will be in vain

8. Final Thoughts

It might seem an inordinate amount of work just to get the front parts of a head into a virtual scene, but the techniques you have learned in this article can be applied to scanning and stitching anything you want. In Tron Legacy the movie, we saw how a human could be scanned so thoroughly that he actually transported his body into a virtual world. The ability to scan your very essence into the
computer and continue your journey in a virtual landscape begins right here with the scanning of a single face. It’s not just scanning a cup or a toy; it’s scanning a precious part of you, and makes the experience very personal indeed.

How long will it be before every computer game ships with the ability to accept the infusion of your personality, preferences and likeness as a common and necessary feature? We could craft these ‘digital doppelgangers’ over many years, and walk them into any game with all the traits we want to share with others. Imagine playing games where almost everyone you meet looks, sounds and reacts just like their human operator. How will games and attitudes change as the in-game protagonists start to resemble real people you might meet in your own life? When you put a face to someone, they become real, and it will be interesting to see how games and players change as a result.

About The Author

When not writing articles, Lee Bamber is the CEO of The Game Creators, a British company that specializes in the development and distribution of game creation tools. Established in 1999, the company and surrounding community of game makers are responsible for many popular brands including Dark Basic, The 3D Game Maker, FPS Creator, App Game Kit (AGK) and most recently, Game Guru.

Notices

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS OTHERWISE AGREED IN WRITING BY INTEL, THE INTEL PRODUCTS ARE NOT DESIGNED NOR INTENDED FOR ANY APPLICATION IN WHICH THE FAILURE OF THE INTEL PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to: http://www.intel.com/design/literature.htm
Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark* and MobileMark*, are measured using specific computer systems, components, software, operations, and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Any software source code reprinted in this document is furnished under a software license and may only be used or copied in accordance with the terms of that license.

Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and/or other countries.

Copyright © 2014 Intel Corporation. All rights reserved.

*Other names and brands may be claimed as the property of others.