Abstract

Documentation on Asset creation for next generation computer game engines
Detailing the processes and software workflows to create a complete digital asset (character) and port it to the Unreal 3 engine

The possibility of my concept will appeal to anyone with a little bit of knowledge about 3d packages and an affinity for games.
More importantly however is the fun aspect of personalising your game. This has been approached previously in a small way such as inserting own logos in "Half-life" etc., but my notion would enable greater recognition and an enhanced user experience.

Introduction

I wanted to find out more about a "next gen" game engine and how to get my own content into them.

Also because I haven't seen it done much before, would be an appealing challenge.

I decided to look at Unreal Tournament because a previous version of the engine had been widely used and I also knew the sdk would be included from website. Plus I believed there would be tutorials on how to use it, in the collector's edition. As stated on their website.
In addition I have personally enjoyed previous versions and thoroughly liked the demo.
Another reason for choosing Unreal was that it already has a large online community in forums and 3rd party websites.

What is "next Gen?"

Next generation is the latest generation of directX10 games. Also the latest Computer Game consoles such as the Xbox 360 and Playstation 3.

What challenges I set myself

I planned to learn how to import a character and textures into the unreal engine editor. Then learn how the editor and included plug-in's work.

Afterwards model and texture from photo reference and sketches, a character in a modular way, similar to that of current Unreal characters. Theoretically enabling the user to mix and match current characters with the new one.

The main priority will be getting the new character and possibly depending on time, a new environment (based on a reconstruction of the Talbot campus) working in game.

How /Why I cut down the scale of the project.
After buying the collectors addition of the game, all of the bonus tutorials where on making levels there were no tutorials on creating characters.
So I decided to keep the scope of the project to a character only.
I thought it would have been good to create my own map, from the experience of the process involved… however decided to do a character because it is more of a challenge. In addition I thought I could learn more by working it out for myself.

How I did – the results

I'm really pleased with my success in achieving the insertion of my digital asset into the "Unreal3" game engine. In addition, I believe the character was a recognizable approximation of my appearance. Also my asset is fully compatible with existing characters and components in unreal tournament.

While the character works I feel that more time could have been spent on creating a slightly more detailed body mesh and higher detailed normal map.

The Documentation or "how to" portion of this report is split into 2 sections the first involving the workflow for creating a useable digital asset. This is using programs that the reader should have a basic understanding of. Therefore some topics will only be talked about briefly. However I still cover this part of the workflow for the sake of thoroughness.

The second section entails the workflow of getting the finished digital asset from Maya and into Unreal Tournament proper. Mostly using the development kit program "UnrealEd version 3.0" and "Notepad" though any text editor will do.

Design and planning the implementation of character

- Quote of interested gamer... "Wouldn't it be great if you could recognise a friend by their own personal game character"?

This would make the game more immersive

- **What should it look like?** Up to you. I decided to try and make my character resemble myself, but not look completely out of place in the game. … However if you want your own animation / radically different character there are certain restrictions that will effect your decisions *(see below).*

After doing initial research from searching UDN developer forums, emailing epic and just playing about with "UnrealEd" opening existing packages and investigating how they work. I found out the following.

Restriction's, if I wanted to use the pre-made animations I would need either a copy of their skeleton, or a similar one with the same naming conventions. So the rotations work correctly.

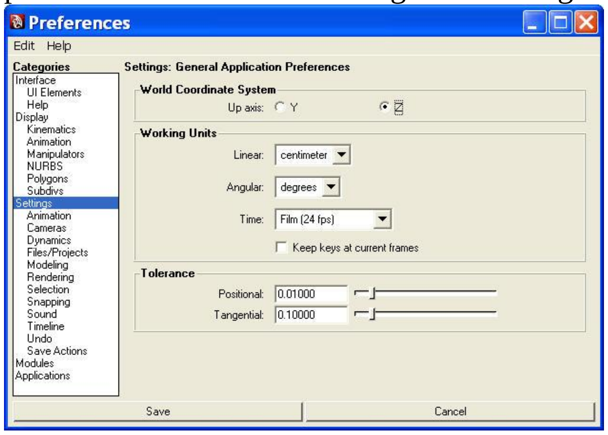
In addition I found that I would need numerous texture maps for the head and body in a 2048x2048 resolution, such as diffuse, normal, specular and emissive maps.

I also found out that the optimal character should only have about a 3000 polygons or 6000 triangle limit for the game to handle it efficiently. It is possible to use more than this, but it isn't advisable. Considering This I decided to put the most detail into the head model, because I thought it was the most important part.

Part 1

Workflow involving "Maya", Photoshop /Gimp and possibly Zbrush and Headus-uvlayout.
Note some illustrations may show the "unreal editor" as a reference but will be covered in greater detail in part 2

Before anything is done in Maya an important setting must be changed. The default world Coordinate UP axis in Maya is the Y axis. Whereas the default "Up" axis in the unreal-engine is the Z axis. Mayas default can be changed by going to the animation preferences menu and selecting these settings:

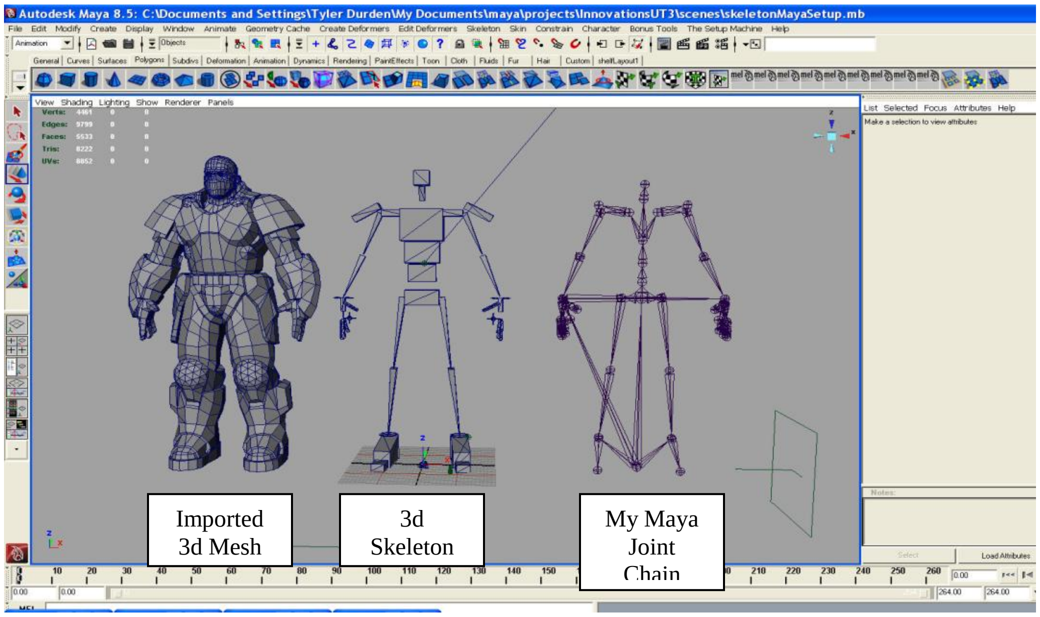

Settings: General Application Preferences

World Coordinate System
Up axis: Y [✓ Z]

Working Units
Linear: centimeter
Angular: degrees
Time: Film (24 fps)
Keep keys at current frames

Tolerance
Positional: 0.01000
Tangential: 0.10000

I created a skeleton from approximating the imported 3dstudiomax file. (This was supplied by Epic games, on the UDN website). By seeing what they had and getting as close to it as possible. As shown below.


Autodesk Maya 8.5: C:\Documents and Settings\Tyler Durden\My Documents\maya\projects\InnovationsUT3\scenes\skeletonMayaSetup.mb
Imported 3d Mesh
3d Skeleton
My Maya Joint Chain

I also had to take into account that as the skeleton is exported to “Unreal” the local rotation axes of the joints are reflected about the y axis. To get this working it involved a **lot** of trial and error. Exporting and importing the skeleton until it worked to a satisfactory level.

To save time I have included a .ma file that contains this working basic skeleton which would save others from having to create it from scratch again. I kept the same number of bones in my skeleton with identical naming conventions. However it turns out while the naming convention is important the number of bones is not. The bones labelled handIK are in-fact unnecessary when using Maya. I left them in just so that the view in the unreal editor would be visually similar and therefore easier for me to comprehend.

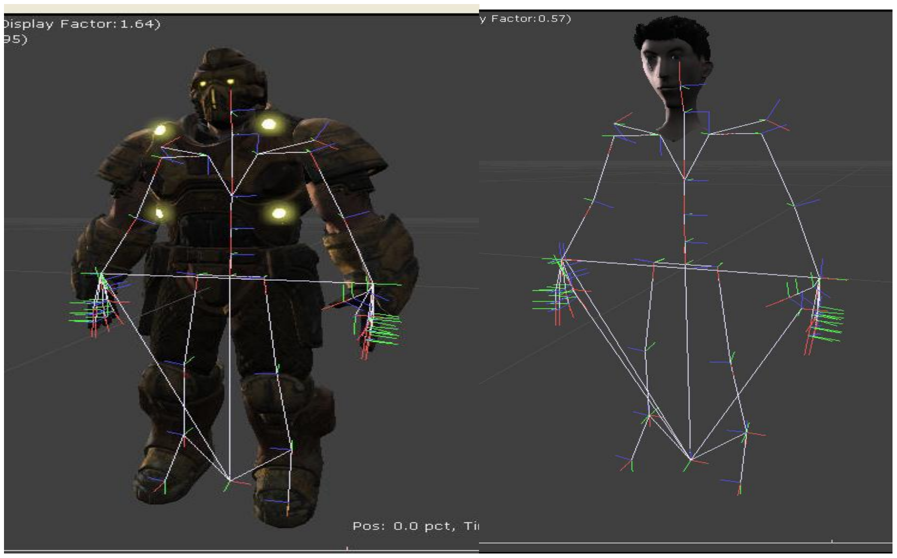
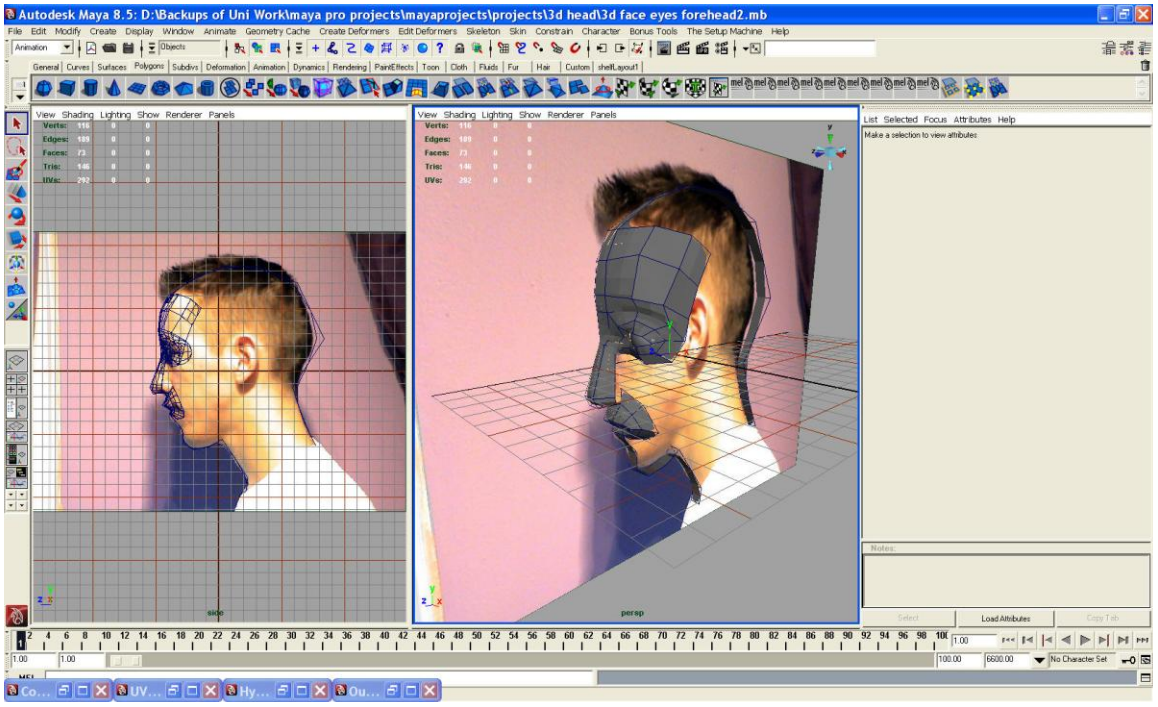


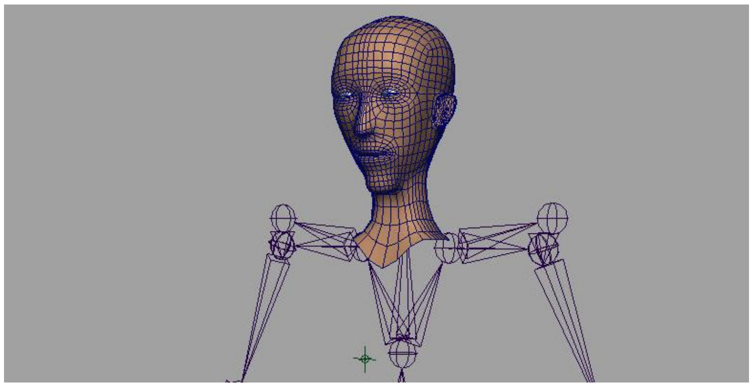
*illustration shows the character and skeleton that was provided by Midway on the left and the skeleton and character head that I created on the left. The white lines depict the skeleton bones and the coloured axes denote the individual joints rotation axes. It was extremely important that these were made to be identical. Otherwise I wouldn't have been able to use the existing animations and would have had to do the 128 or so separate animation takes for my character, again from scratch myself.

Modelling

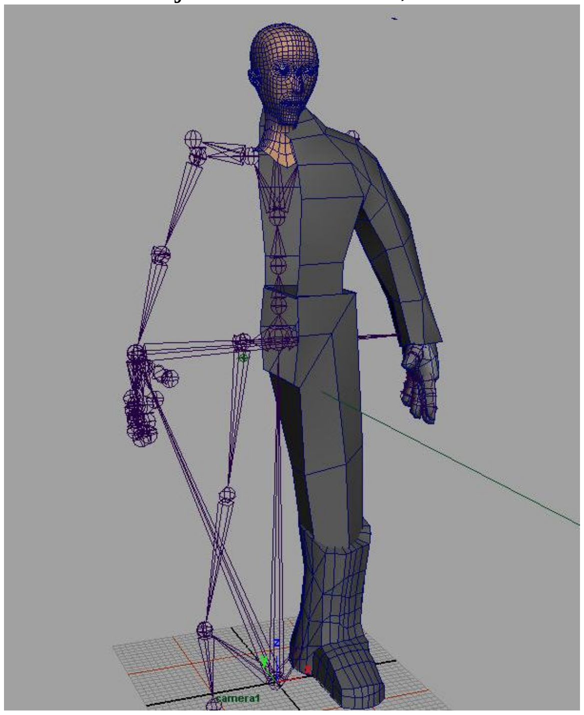
It helps if you have good reference images and or model sheets to work from. However in this case it is beneficial to model your character to the skeleton, rather than creating a skeleton, or modifying the skeleton to fit your character. Or basically make sure your model sheets are scaled to the size of the skeleton that has been created earlier.


Autodesk Maya 8.5: D:\Backups of Uni Work\maya pro projects\mayaprojects\projects\3d head\3d face eyes forehead2.mb
File Edit Modify Create Display Window Animate Geometry Cache Create Deformers Edit Deformers Skeleton Skin Constrain Character Bonus Tools The Setup Machine Help
View Shading Lighting Show Renderer Panels
side
persp
List Selected Focus Attributes Help
Make a selection to view attributes
Load Attributes

*the picture shows the Maya interface and the front and side views of my model. Building up the head mesh using a polygon proxy on a subdivision surface. Extending out from the brow of the nose outwards and creating the eyes and mouth. The rest of the head model is then extruded out from these details. When the model looks satisfactory, it then needs to be converted into polygons for later.


*picture is illustrating the finished head model with clean topology and basic skin colour.

Since I had the head modelled to a satisfactory level I began to model a basic body shape trying to keep the proportions similar to the existing “unreal” model. I create and work on half of the object then mirror it, to create a full mesh faster.

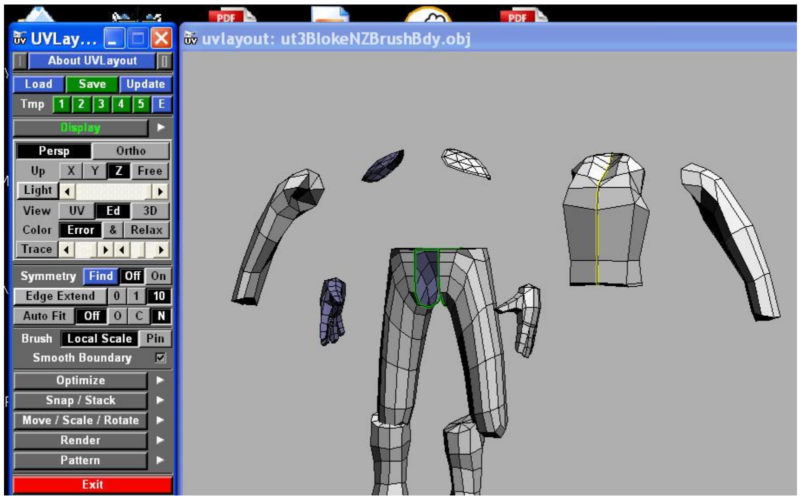


This illustration shows How I began to block out the model, keeping the polygon count down. The other half is created from a mirror image so it keeps it symmetrical. Also shows the Skeleton that the model needs to be fitted too.

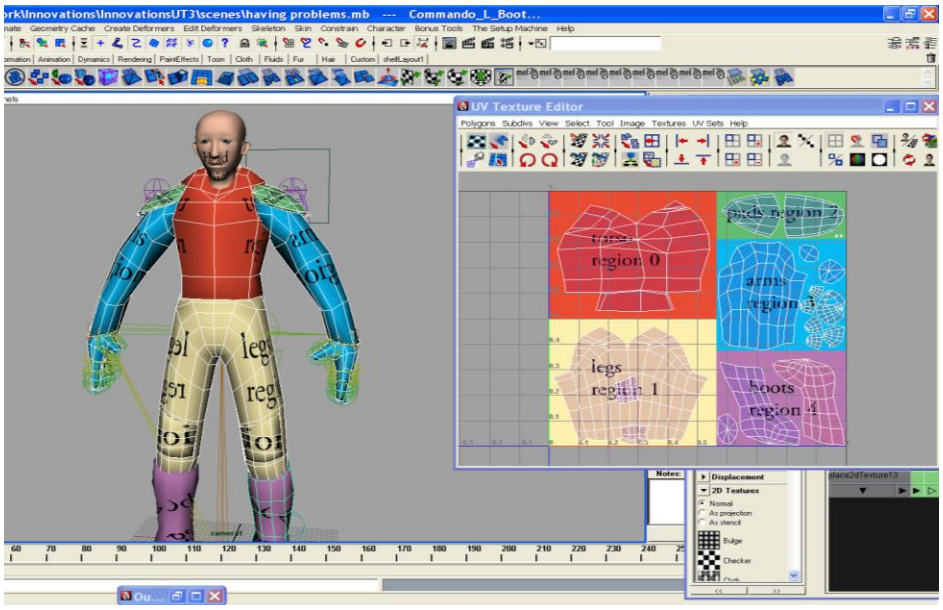
UV layout

It’s important to get the UV regions of different parts of the model in the correct place; otherwise this could create problems further down the line. Such as when trying to integrate parts of your digital asset with the pre-made in game parts such as helmets and goggles etc. Keeping your models simple will greatly speed up this process as this will make flattening the Uv maps a lot easier. I find that some 3rd party applications such as Headus Uv layout can help to layout Uvs more efficiently than the built in tools in Maya.

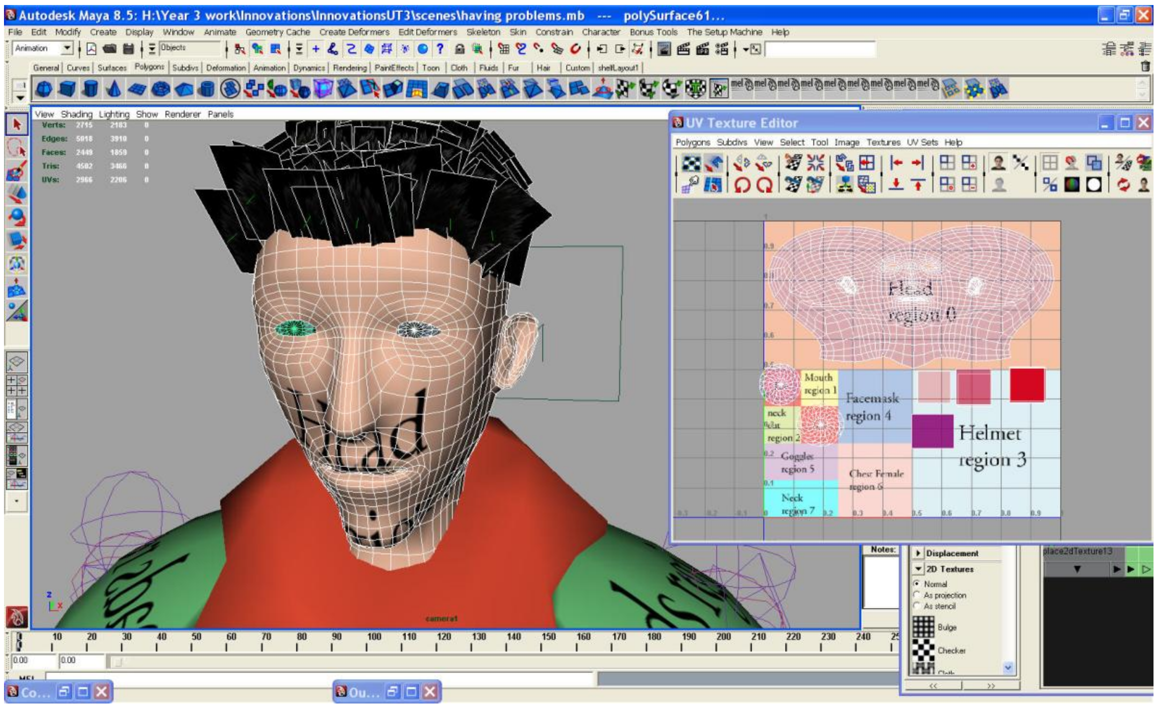
The illustration below is of the Headus UV Layout program, It shows the interface and the ease in which the model can be separated into the correct parts, to make more comprehensible uv layouts / texture maps



Here is how I laid out my characters uvs in Maya.



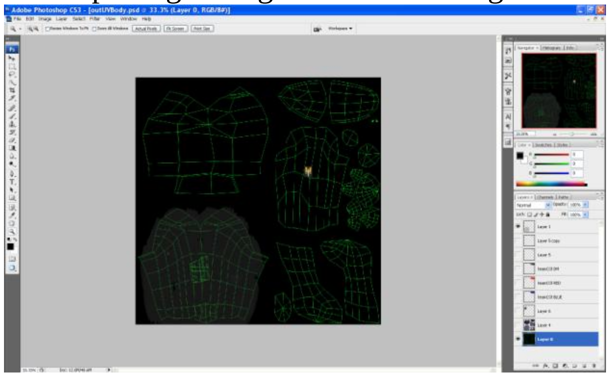
As you can see each coloured section on the map corresponds to certain body parts.
I did this for 1 side of the model then mirrored it so both side share the same UV space.
This is done to make the textures more efficient. However in for objects such as the torso and head it's a better idea to layout the uvs for the entire object



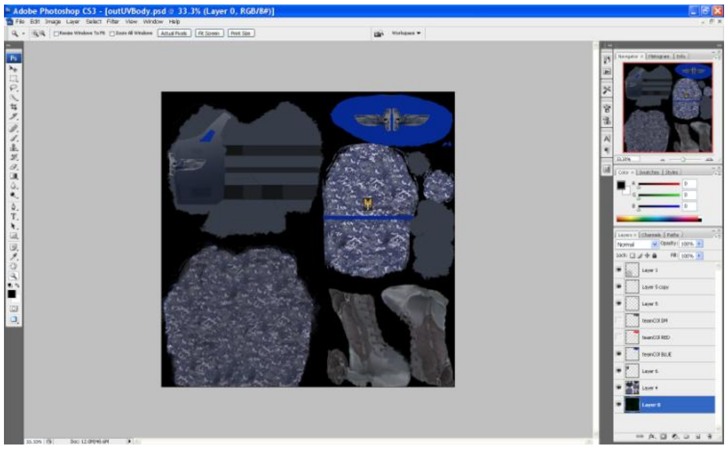
This picture shows the head uv layout, the hair geometry is a collection of quads with a texture mapped onto them. The uv coordinates of the hair are in part of the helmet section because one of the default unreal characters had a similar arrangement.

Texturing

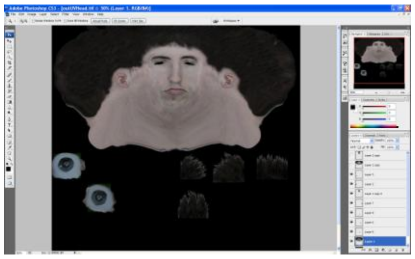
Since I laid out the uv's earlier I used this as a basis to create textures for the character. This is done by using the UVSnapshot Tool in the UV Texture Editor in Maya.
To keep things straightforward making the size is 2048 x 2048 and saved as a .tiff file.


Adobe Photoshop CS3 - [outUVBody.psd @ 33.3% (Layer 0, RGB/8#)]

Texturing the body.
Now I wanted a military feel to my character, so I found an example camouflage picture and used it as reference for the main colour I Also used references images for the vest and boots. I would usually take a lot more time and make much higher detailed texture. However at this point I was unsure if I could complete the whole project in the timeframe. I also believed as long as I managed to get the head and face to a good likeness that would be enough. Therefore I left it at this level of detail. I can always go back and work more detail into the body texture at a later date.



Once this step was completed I then used the original head reference images in conjunction with the UV-snapshots from Maya to create the head hair and eye textures. As shown. Once you have created a texture save it as a TGA image file of dimensions 2048x 2048



Normal Mapping

This step **isn't** essential, so if you want to just get a basic model into unreal you can skip it entirely. However it can be fundamental in getting models that look of a very high quality, working within the "Unreal" engine.

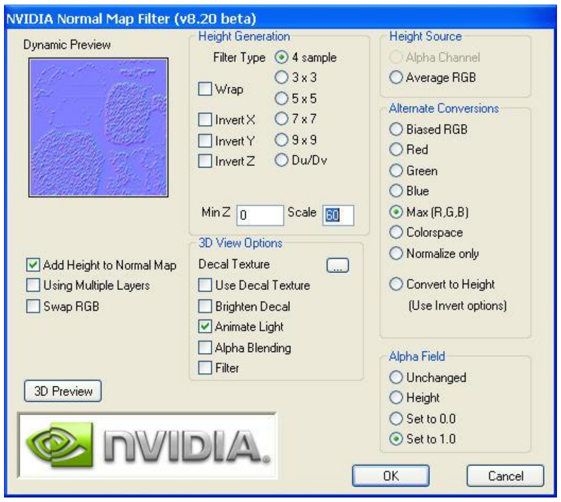
What is Normal mapping?

Quoted reference for Normal Mapping taken from wikipedia.org

"…**normal mapping** is an application of the technique known as bump mapping. While bump mapping perturbs the existing normal (the way the surface is facing) of a model, normal mapping replaces the normal entirely. Like bump mapping, it is used to add details to shading without using more polygons. But where a bump map is usually calculated based on a single-channel (interpreted as grayscale) image, the source for the normal's in normal mapping is usually a multi-channel image (x, y and z channels) derived from a set of more detailed versions of the objects. The values of each channel usually represent the xyz coordinates of the normal in the point corresponding to that Texel…"

I used "Zbrush3.0" to create my Normal map. I have found that while certainly not being the easiest way to do so. If done correctly it can achieve the most detail as an end result. (Zbrush can handle over 1Million polygons on a single mesh. I subdivided the standard mesh from Maya up to around 650,000 polygons and then sculpted directly onto that mesh.)

However if you do not have access to Zbrush there are many other ways to create normal maps, such as: NVIDIA normal map filter for adobe Photoshop creates a normal map from a diffuse texture... (Easiest but the least controllable/accurate). As shown below


NVIDIA Normal Map Filter (v8.20 beta)

Dynamic Preview

Height Generation
Filter Type: [● 4 sample] ○ 3 x 3 ○ 5 x 5 ○ 7 x 7 ○ 9 x 9 ○ Du/Dv
☐ Wrap
☐ Invert X
☐ Invert Y
☐ Invert Z
Min Z: 0
Scale: 60

Height Source
○ Alpha Channel
○ Average RGB

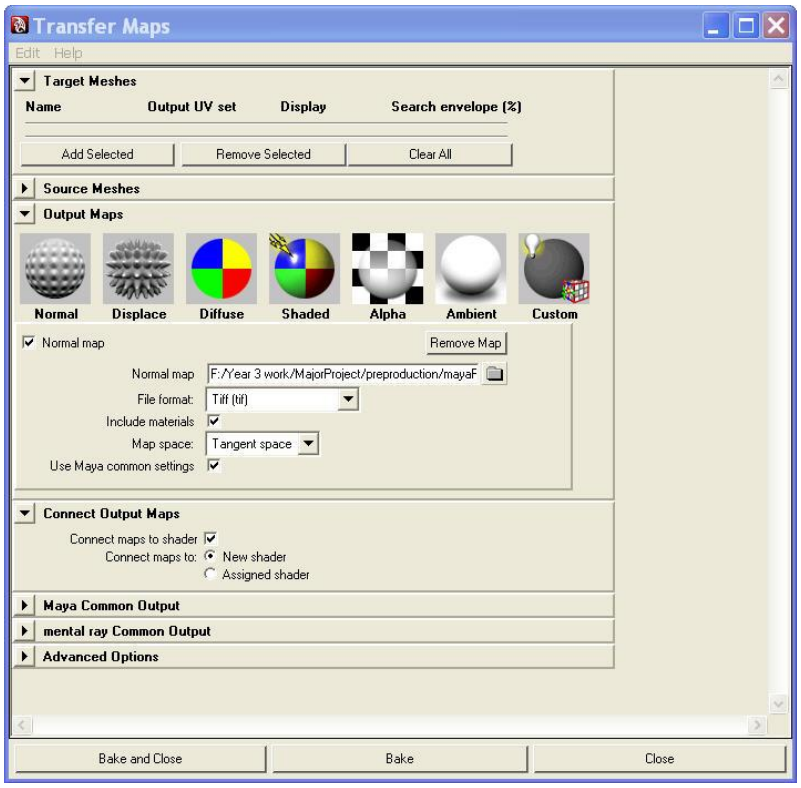
Alternate Conversions
○ Biased RGB
○ Red
○ Green
○ Blue
[● Max (R,G,B)]
○ Colorspace
○ Normalize only
○ Convert to Height (Use Invert options)

☑ Add Height to Normal Map
☐ Using Multiple Layers
☐ Swap RGB

3D View Options
Decal Texture
☐ Use Decal Texture
☐ Brighten Decal
☑ Animate Light
☐ Alpha Blending
☐ Filter

Alpha Field
○ Unchanged
○ Height
○ Set to 0.0
[● Set to 1.0]

An additional way is to create a copy of your mesh in Maya and work much more detail into it (more subdivisions). Then use the transfer maps tool in Maya.


Transfer Maps

Target Meshes

Name	Output UV set	Display	Search envelope (%)

Source Meshes

Output Maps

☑ Normal map

Normal map: F:/Year 3 work/MajorProject/preproduction/mayaF
File format: Tiff (tif)
Include materials: ☑
Map space: Tangent space
Use Maya common settings: ☑

Connect Output Maps
Connect maps to shader: ☑
Connect maps to: [● New shader] ○ Assigned shader

This creates a normal map from the high res mesh onto the low res object... (Slightly more time consuming but can give a very good result with the correct settings.

*note using this method the ray-tracing calculations can take over an hour for a 2048x2048 map on high settings. Also Maya Does have a tendency to crash or be very very slow, when the high res object is over 200,000 polygons. This obviously depends on the specifications of your machine.)

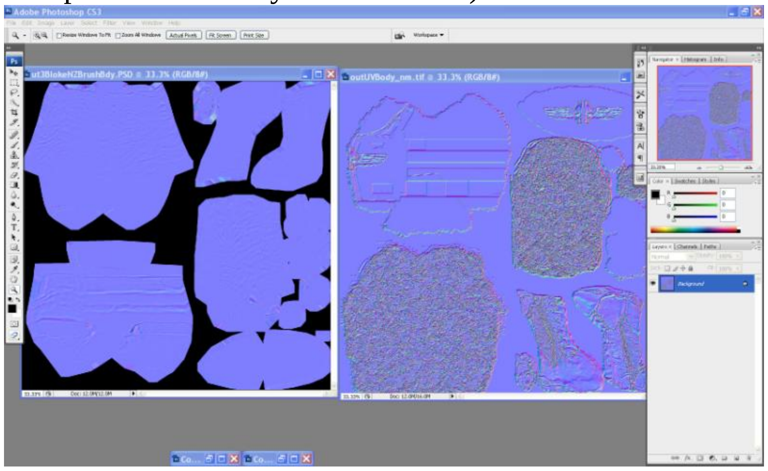
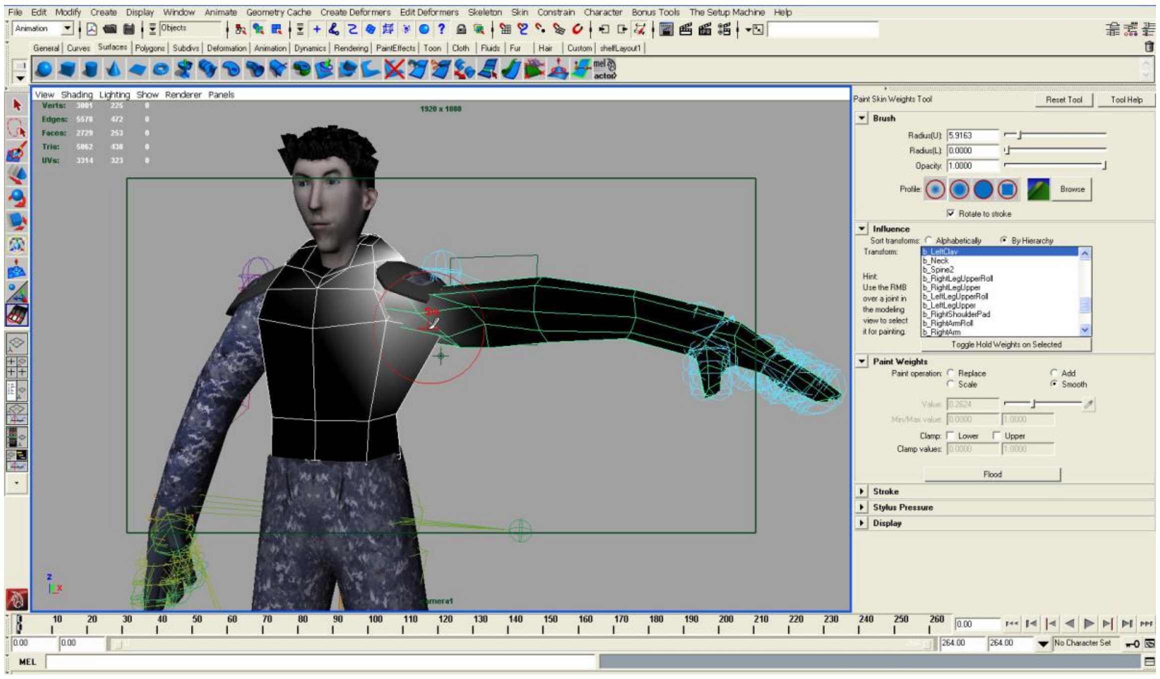

Adobe Photoshop CS3
ut3BlokeNZBrushBdy.PSD @ 33.3% (RGB/8#)
outUVBody_nm.tif @ 33.3% (RGB/8#)

Image shows the Zbrush normal map beside NVIDIA's normal map filter.

Both have their advantages but they can also be blended together to create an even better result

Rigging the character

Select all joints of the skeleton and all of the geometry of the mesh and chose smooth bind with the default settings. Then need to paint weights on the individual parts, so when you move the skeleton it deforms the geometry correctly. For example the joints on the arms only affect the arm geometry etc. I have to undo any joint rotations before tweaking or changing the weighting of the mesh, because the "assume preferred angle" command on the skeleton, will **not** work correctly. It was set the way when I created the skeleton hierarchy and tweaked it, so it will work with the default "Unreal" character animations (*the local rotation axes are not "frozen" or zeroed).



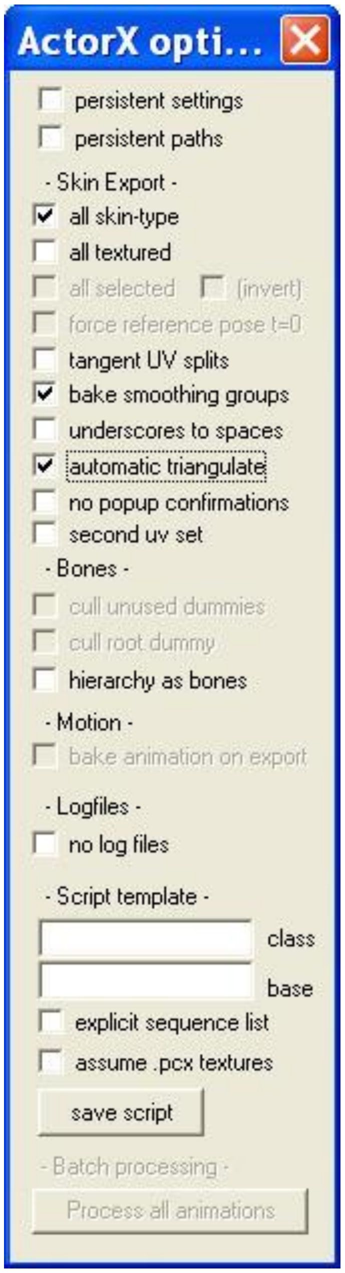
The above illustration shows an example of how to paint weights correctly on the chest and arm of my character.

If you just want to use the default animations that are in the game engine already. Then there is no need to add IK controls or constraints, the given skeleton just has to move the mesh in the desired way from rotational information only.

It is a good idea to thoroughly test it **before** exporting; this saves a great deal of time later. Hence you don't have to re-export everything from "Maya" and re-import everything back into "UnrealEd" more than once. Which I unfortunately had to do on numerous occasions...

Exporting the finished character

Now the finished asset has to be exported from Maya into a format the Unreal Editor can understand. To help with this thankfully there is a plug-in for Maya to handle this. It is called the actorX plug-in and is also available from the UDN developer website. The actor X plug-in has to be activated like any other in the settings and preferences menu in Maya. Once it's been activated, the Mel command "axmain" will run the main actor x script. Before you run the main command its worth running "axoptions" make sure these settings are enabled.


ActorX opti...

☐ persistent settings
☐ persistent paths

- Skin Export -
☑ all skin-type
☐ all textured
☐ all selected ☐ (invert)
☐ force reference pose t=0
☐ tangent UV splits
☑ bake smoothing groups
☐ underscores to spaces
☑ automatic triangulate
☐ no popup confirmations
☐ second uv set

- Bones -
☐ cull unused dummies
☐ cull root dummy
☐ hierarchy as bones

- Motion -
☐ bake animation on export

- Logfiles -
☐ no log files

- Script template -
class
base
☐ explicit sequence list
☐ assume .pcx textures

save script

- Batch processing -

Process all animations

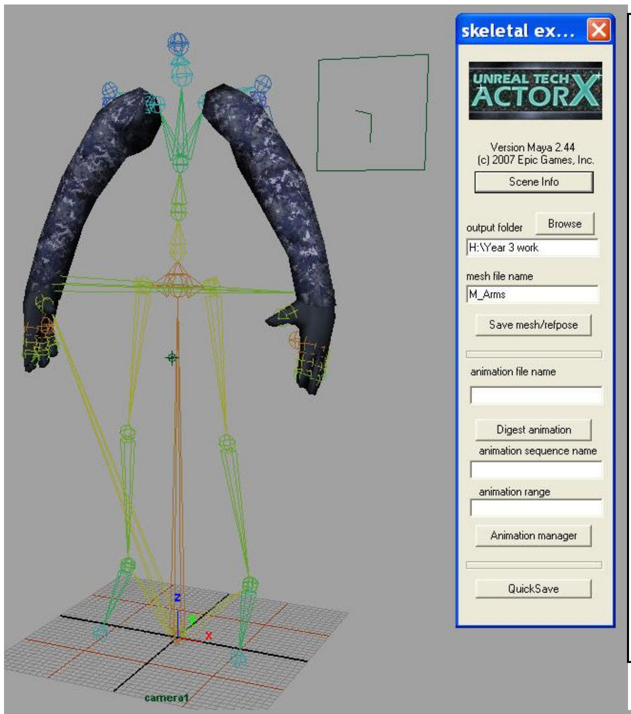
This illustration shows the correct settings for the “axoptions” command.

“all skin-type” basically means it exports any mesh that is rigged to a skeletal chain.

The automatic triangulate is very important if you have modelled your character using quads (4 sided polygons) because the unreal engine needs triangulated meshes to work correctly and will produce an error when trying to import a mesh that has polygons of 4 sides or more. You may want to triangulate the mesh yourself to produce a better result.

Unfortunately the “cull unused dummies” and “cull root dummy” options do not work for Maya 8.5. This makes the next step slightly more time consuming, but not terribly so.

Because of the way that the ActorX plug-in works with Maya 8.5, it is necessary to **delete** all the objects bar the skeleton and the individual part that you require. **Before deleting anything. Save a separate copy of the scene file, to refer back to. This enables us to have separate body “parts” in game, which can be used in conjunction with the default parts that are already included in game. It is possible to skip this step and just export the entire mesh at once. However you wouldn’t get the same integration with default parts.



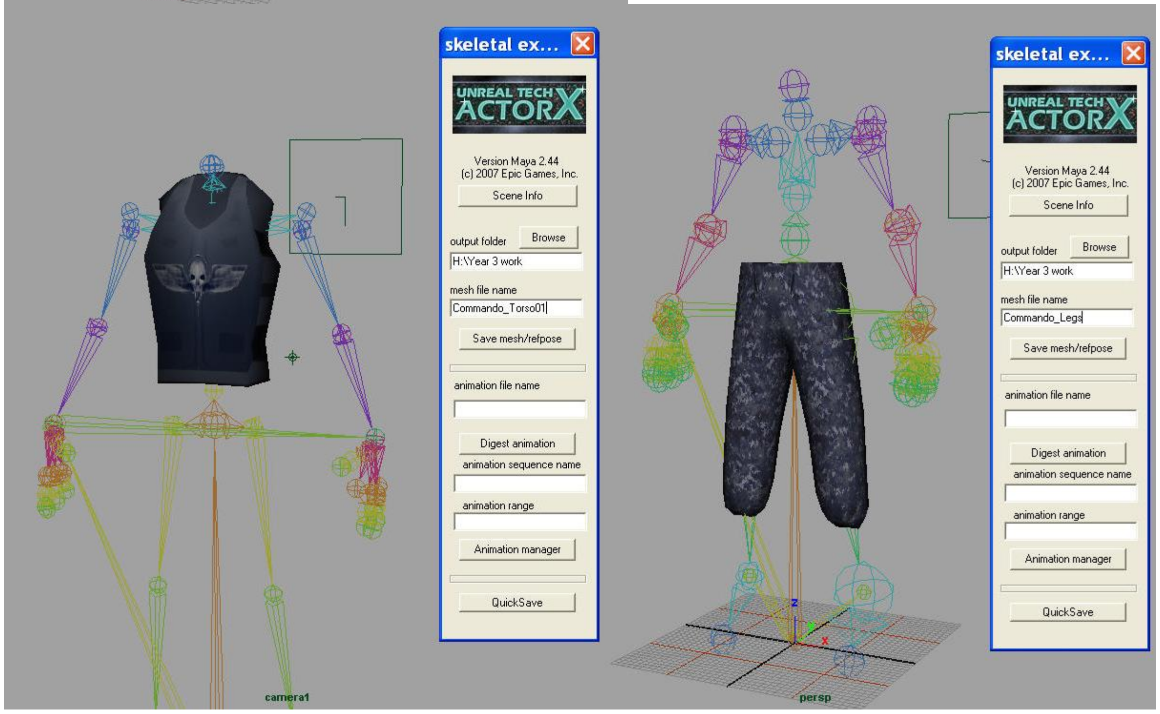
output folder: H:\Year 3 work

mesh file name: M_Arms

Illustration shows how to export the hands and arms. By deleting the other parts. Naming the parts [packageName]_Arms.

Then click the “save mesh/refpose” Button

I saved the parts to a separate folder to keep things simple



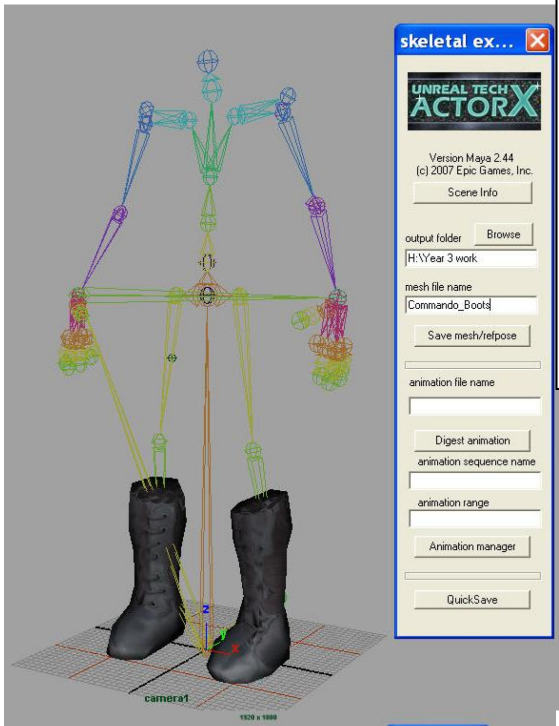
output folder: H:\Year 3 work

mesh file name: Commando_Torso01

output folder: H:\Year 3 work

mesh file name: Commando_Legs

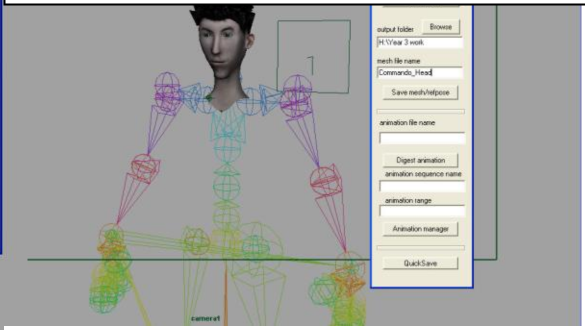
Illustration shows how to export the torso with name [pakagename]_Torso01
And how to export the legs with the name [packageName]_Legs01



output folder H:\Year 3 work

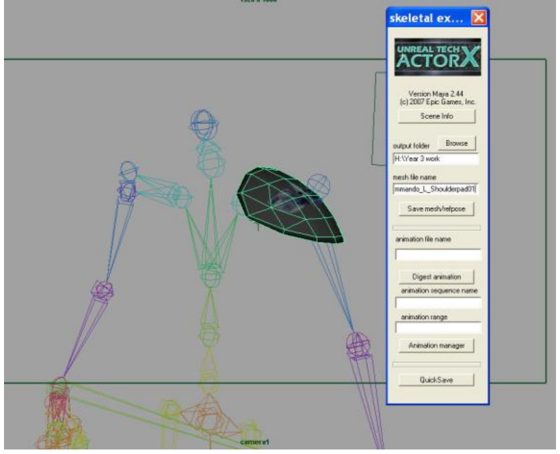
mesh file name Commando_Boots

Illustration shows how to export the Boots with name [pakagename]_Boots01
And how to export the head with the name [packageName]_Head01



output folder H:\Year 3 work

mesh file name Commando_Head



output folder H:\Year 3 work

mesh file name mmando_L_Shoulderpad01

Illustration shows how to export the Left shoulderpad with name [packageName]_L_ShoPad01

The same also has to be done for the right shoulder pad its name should read [packageName]_R_Shopad01

I had to repeat the previous steps numerous times, before I managed to get to the final result.

The naming conventions are important but not essential to making it work. It just makes scripting and general use with high numbers of parts easier to comprehend.

Hopefully this report will help anyone else with doing the something similar, with ease.

End of part1

Part 2 UNREAL workflow

Open the Unreal editor by default for windows users it should be in: start->all programs -> unreal Tournament 3 -> unreal tournament editor

Welcome to the main unreal editor, for creating custom levels this is your main interface. However while creating a character you actually won't be using this interface screen at all really. Firstly what needs to be done is to open the generic browser window. In view menu -> Browser Windows -> generic

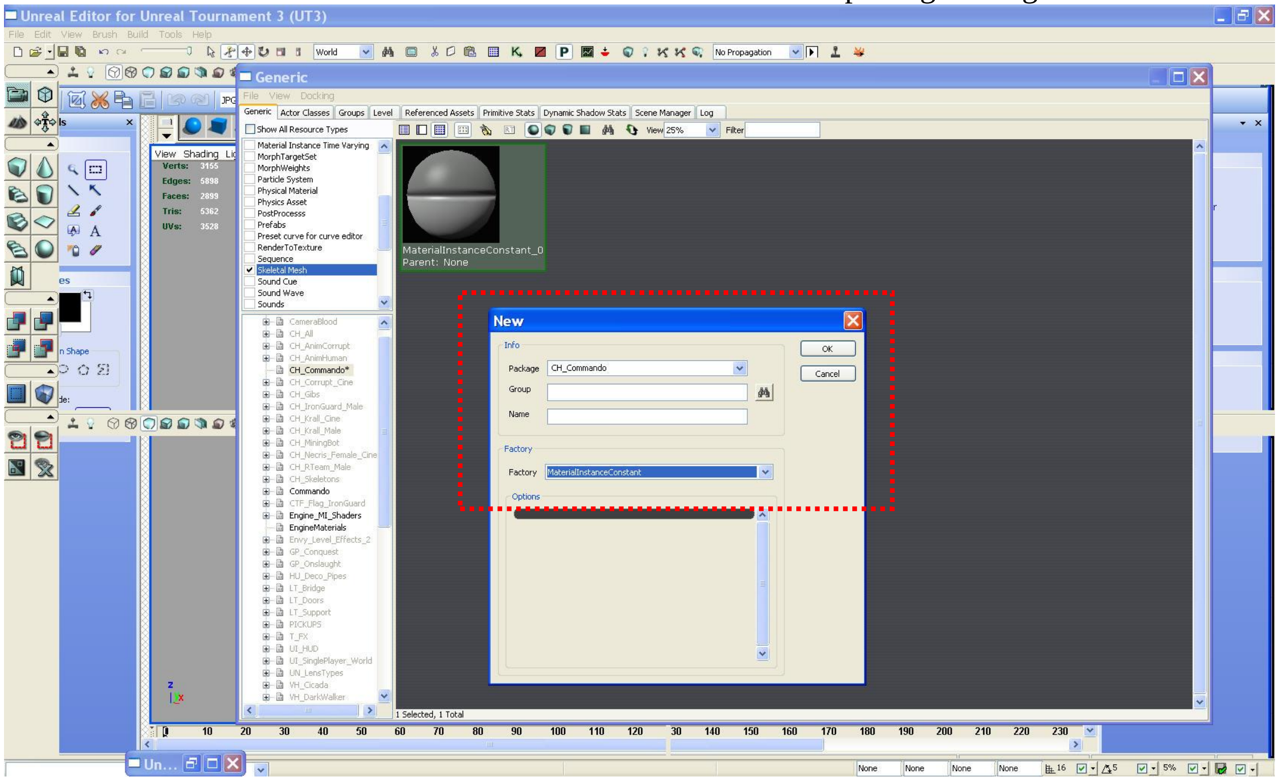
Making a new package

The unreal engine works on packages; these are basically collections of assets for example textures models and code etc.

So here is how to create your container package.

Also this shows how to create the first **materia**l for our character.

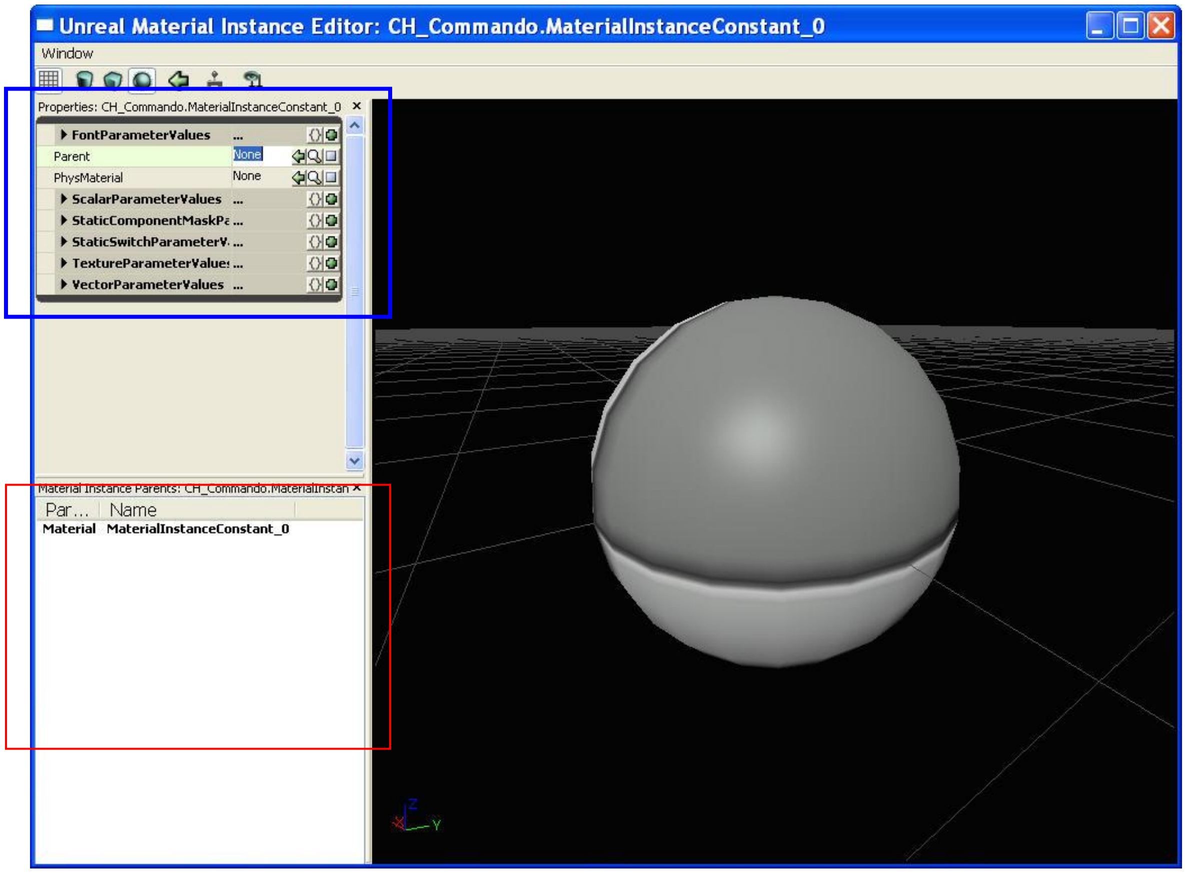
This illustration shows the Generic browser window and the new package dialogue


Unreal Editor for Unreal Tournament 3 (UT3)
File Edit View Brush Build Tools Help
Generic
Generic Actor Classes Groups Level Referenced Assets Primitive Stats Dynamic Shadow Stats Scene Manager Log
Show All Resource Types
MaterialInstanceConstant_0
Parent: None
1 Selected, 1 Total
New
OK
Cancel

Info
Package CH_Commando
Group
Name
Factory
Factory MaterialInstanceConstant
Options

In the New package dialogue Type in your new package name in my case it was "Commando". Under groups the new group should be called "Materials" The name should be MIC_[packageName]_Body_V01

The Factory type should be set to Material Instance Constant. (MIC for short). This is the shader/material for the objects like in the hypershade in Maya; it contains instructions to display textures.
When ok is clicked, you get a new swatch in the generic browser, Double click on the new MIC to display the "**material instance editor**"


Unreal Material Instance Editor: CH_Commando.MaterialInstanceConstant_0
Window

Properties: CH_Commando.MaterialInstanceConstant_0

FontParameterValues	...
Parent	None
PhysMaterial	None
ScalarParameterValues	...
StaticComponentMaskP…	...
StaticSwitchParameterV…	...
TextureParameterValue…	...
VectorParameterValues	...

Material Instance Parents: CH_Commando.MaterialInstan…

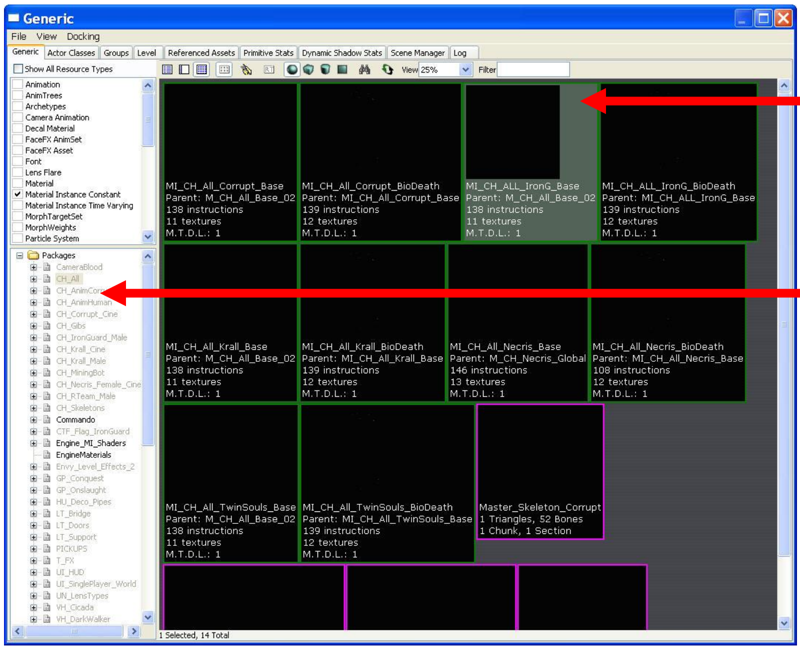
Par…	Name
Material	MaterialInstanceConstant_0

Since this MIC is new. It is completely blank, it displays the default grey checker texture, but since the MIC is blank. It has no instructions to display textures or other functions.

Now there is no reason to "re-invent the wheel" by setting these instructions up from scratch. They can be inherited from a Base MIC class (that the nice people at Midway have provided)

By setting the Base MIC as a parent, our MIC will inherit all the instructions and base textures
So in the blue section above we need it to read the location of the BASE MIC.

So while leaving this editor window open, in the Generic browser window, select the CH_ALL package as in the picture below. Numerous MIC will be visible. Since my characters Skeleton is of the IRONGUARD_MALE makes sense to use the **MI_CH_ALL_IRONG_BASE** Material. Select it by clicking it once. So it will highlight as in the picture.


Generic
File View Docking
Generic Actor Classes Groups Level Referenced Assets Primitive Stats Dynamic Shadow Stats Scene Manager Log
Show All Resource Types
View 25%
Filter

Animation
AnimTrees
Archetypes
Camera Animation
Decal Material
FaceFX AnimSet
FaceFX Asset
Font
Lens Flare
Material
✓ Material Instance Constant
Material Instance Time Varying
MorphTargetSet
MorphWeights
Particle System

Packages
CameraBlood
CH_All
CH_AnimCorr
CH_AnimHuman
CH_Corrupt_Cine
CH_Gibs
CH_IronGuard_Male
CH_Krall_Cine
CH_Krall_Male
CH_MiningBot
CH_Necris_Female_Cine
CH_RTeam_Male
CH_Skeletons
Commando
CTF_Flag_IronGuard
Engine_MI_Shaders
EngineMaterials
Envy_Level_Effects_2
GP_Conquest
GP_Onslaught
HU_Deco_Pipes
LT_Bridge
LT_Doors
LT_Support
PICKUPS
T_FX
UI_HUD
UI_SinglePlayer_World
UN_LensTypes
VH_Cicada
VH_DarkWalker

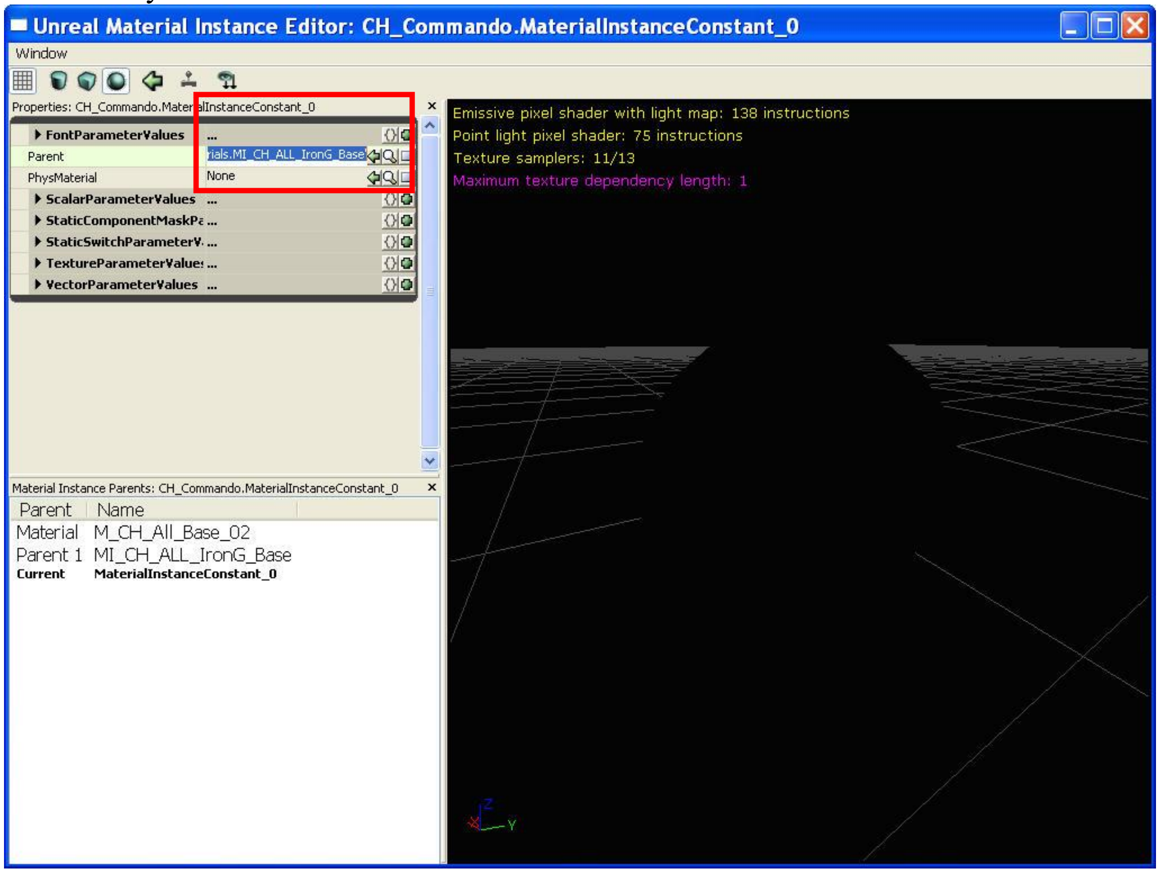
```
MI_CH_All_Corrupt_Base          MI_CH_All_Corrupt_BioDeath        MI_CH_ALL_IronG_Base           MI_CH_ALL_IronG_BioDeath
Parent: M_CH_All_Base_02        Parent: MI_CH_All_Corrupt_Base    Parent: M_CH_All_Base_02       Parent: MI_CH_ALL_IronG_Base
138 instructions                139 instructions                  138 instructions               139 instructions
11 textures                     12 textures                       11 textures                    12 textures
M.T.D.L.: 1                     M.T.D.L.: 1                       M.T.D.L.: 1                    M.T.D.L.: 1

MI_CH_All_Krall_Base            MI_CH_All_Krall_BioDeath          MI_CH_All_Necris_Base          MI_CH_All_Necris_BioDeath
Parent: M_CH_All_Base_02        Parent: MI_CH_All_Krall_Base      Parent: M_CH_Necris_Global     Parent: MI_CH_All_Necris_Base
138 instructions                139 instructions                  146 instructions               108 instructions
11 textures                     12 textures                       13 textures                    12 textures
M.T.D.L.: 1                     M.T.D.L.: 1                       M.T.D.L.: 1                    M.T.D.L.: 1

MI_CH_All_TwinSouls_Base        MI_CH_All_TwinSouls_BioDeath      Master_Skeleton_Corrupt
Parent: M_CH_All_Base_02        Parent: MI_CH_All_TwinSouls_Base  1 Triangles, 52 Bones
138 instructions                139 instructions                  1 Chunk, 1 Section
11 textures                     12 textures
M.T.D.L.: 1                     M.T.D.L.: 1
```

1 Selected, 14 Total

This is the base material **MI_CH_ALL_IRONG_BASE** It should be highlighted as shown.

This is the package list, most of the packages that are used in the game can be found here.
The base material we are looking for is in the CH_ALL Package.

Now in the Material Instance editor that is open select the "use current selection in browser key" as shown below.


Unreal Material Instance Editor: CH_Commando.MaterialInstanceConstant_0
Window

Properties: CH_Commando.MaterialInstanceConstant_0

FontParameterValues	...
Parent	rials.MI_CH_ALL_IronG_Base
PhysMaterial	None
ScalarParameterValues	...
StaticComponentMaskPa	...
StaticSwitchParameterV	...
TextureParameterValue	...
VectorParameterValues	...

```
Emissive pixel shader with light map: 138 instructions
Point light pixel shader: 75 instructions
Texture samplers: 11/13
Maximum texture dependency length: 1
```

Material Instance Parents: CH_Commando.MaterialInstanceConstant_0

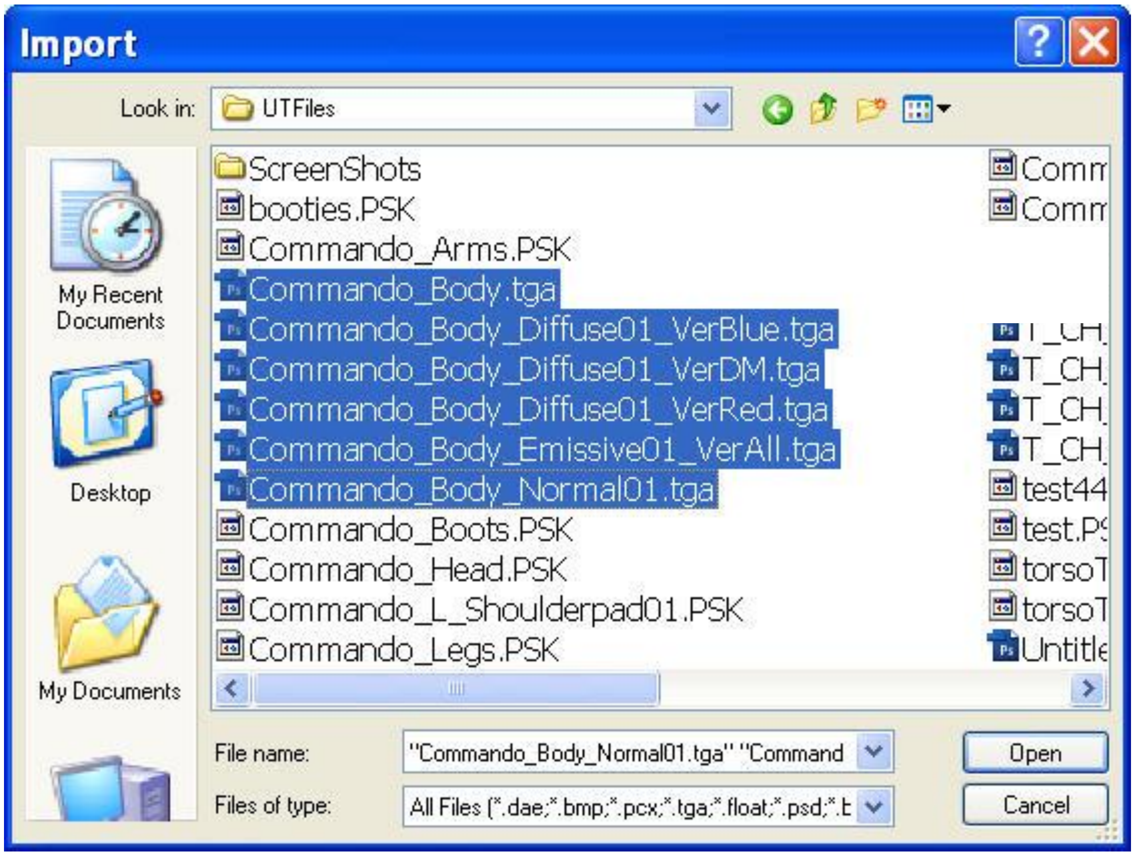
Parent	Name
Material	M_CH_All_Base_02
Parent 1	MI_CH_ALL_IronG_Base
Current	**MaterialInstanceConstant_0**

Our Mic will turn black (because we haven't applied a texture yet!). It will also state that our MIC now has some instructions.

Importing textures

Now the textures that were created in Photoshop earlier need to be imported.

To do this select our package, “commando” in my case and in the file menu select import. The following dialogue box should appear.


Import
Look in:
UTFiles
My Recent Documents
Desktop
My Documents
ScreenShots
booties.PSK
Commando_Arms.PSK
Commando_Body.tga
Commando_Body_Diffuse01_VerBlue.tga
Commando_Body_Diffuse01_VerDM.tga
Commando_Body_Diffuse01_VerRed.tga
Commando_Body_Emissive01_VerAll.tga
Commando_Body_Normal01.tga
Commando_Boots.PSK
Commando_Head.PSK
Commando_L_Shoulderpad01.PSK
Commando_Legs.PSK
Open
Cancel

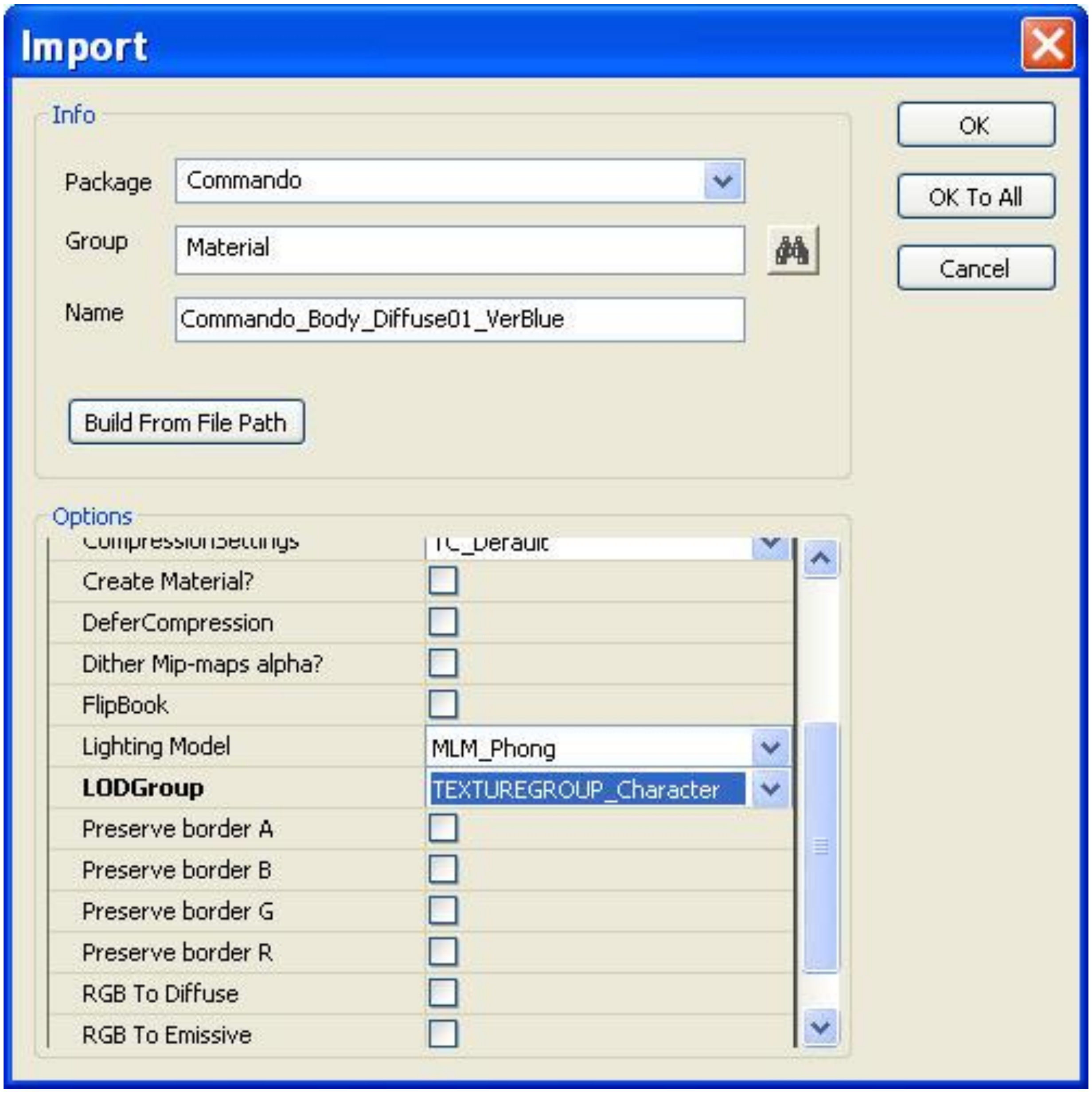
File name: "Commando_Body_Normal01.tga" "Command

Files of type: All Files (*.dae;*.bmp;*.pcx;*.tga;*.float;*.psd;*.t

Select all the created textures diffuse, emissive, specular, opacity and normal for both the head and body if you have them.

I added my specular map after this picture was taken, so it isn’t imperative to have everything at this point, its possible add things later, it just makes sense to import them all at once to save time.

Once they are opened the Import dialogue box will open. As shown below



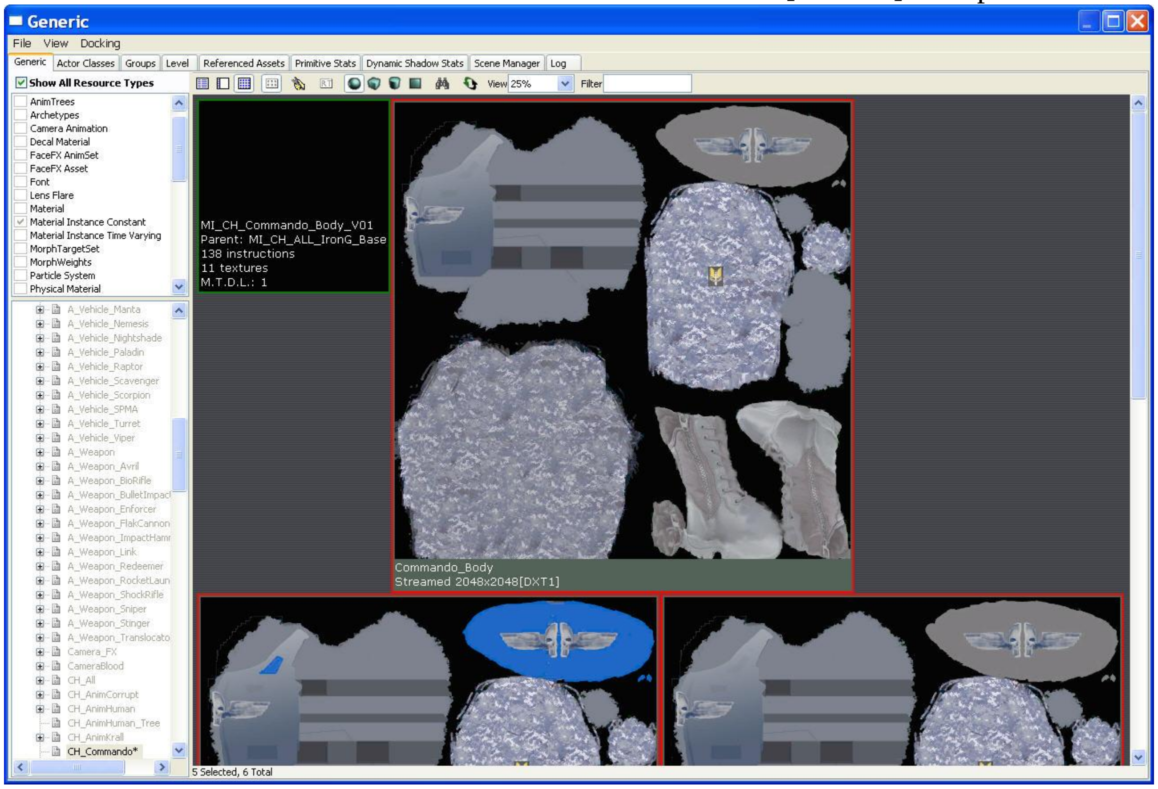
Package: Commando
Group: Material
Name: Commando_Body_Diffuse01_VerBlue

CompressionSettings	TC_Default
Create Material?	☐
DeferCompression	☐
Dither Mip-maps alpha?	☐
FlipBook	☐
Lighting Model	MLM_Phong
LODGroup	TEXTUREGROUP_Character
Preserve border A	☐
Preserve border B	☐
Preserve border G	☐
Preserve border R	☐
RGB To Diffuse	☐
RGB To Emissive	☐

The package name should read the package you want to import into. The group should be material. However groups are just to organise things, for our benefit but their names will make a difference later, when scripting. The name should be identical to the texture name. The LOD Group should be TEXTUREGROUP_Character for diffuse, TEXTUREGROUP_CharacterNormal for normal maps and so on.

Once the ok button is clicked the “Unreal ed” takes our TGA textures and converts them into a proprietary format that the engine can read. This process can take quite some time especially with large numbers of textures.

Once they are loaded the textures should show up in the Generic browser in red boxes with a name and beneath it should read steamed 2048x 2048 [DXT10] as depicted below.

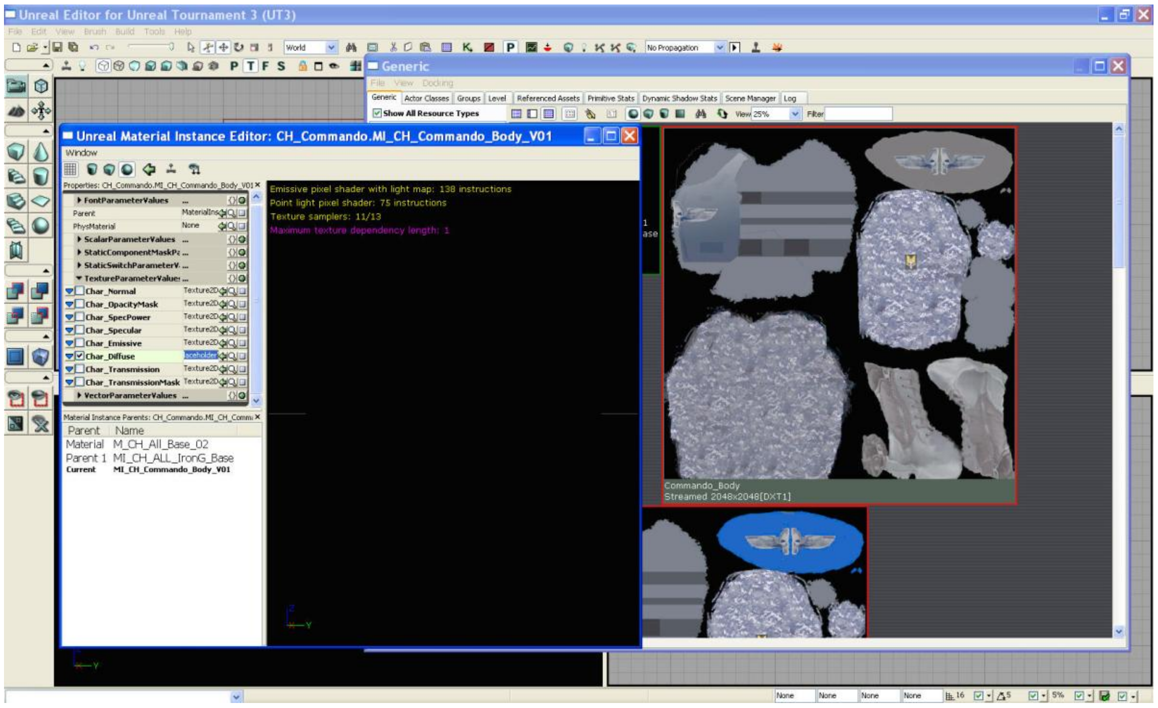

Generic
File View Docking
Generic Actor Classes Groups Level Referenced Assets Primitive Stats Dynamic Shadow Stats Scene Manager Log
Show All Resource Types
View 25%
Filter

```
MI_CH_Commando_Body_V01
Parent: MI_CH_ALL_IronG_Base
138 instructions
11 textures
M.T.D.L.: 1
```

```
Commando_Body
Streamed 2048x2048[DXT1]
```

5 Selected, 6 Total

Now we have to add these textures to Our MIC we created earlier. Since we created the MIC for the Body we need to highlight the body texture in the generic browser window as shown.


Unreal Editor for Unreal Tournament 3 (UT3)
Unreal Material Instance Editor: CH_Commando.MI_CH_Commando_Body_V01
Window
Generic

Properties: CH_Commando.MI_CH_Commando_Body_V01

FontParameterValues	...
Parent	MaterialIns
PhysMaterial	None
ScalarParameterValues	...
StaticComponentMaskP	...
StaticSwitchParameterV	...
TextureParameterValue	...
Char_Normal	Texture2D
Char_OpacityMask	Texture2D
Char_SpecPower	Texture2D
Char_Specular	Texture2D
Char_Emissive	Texture2D
✓ Char_Diffuse	aceholder
Char_Transmission	Texture2D
Char_TransmissionMask	Texture2D
VectorParameterValues	...

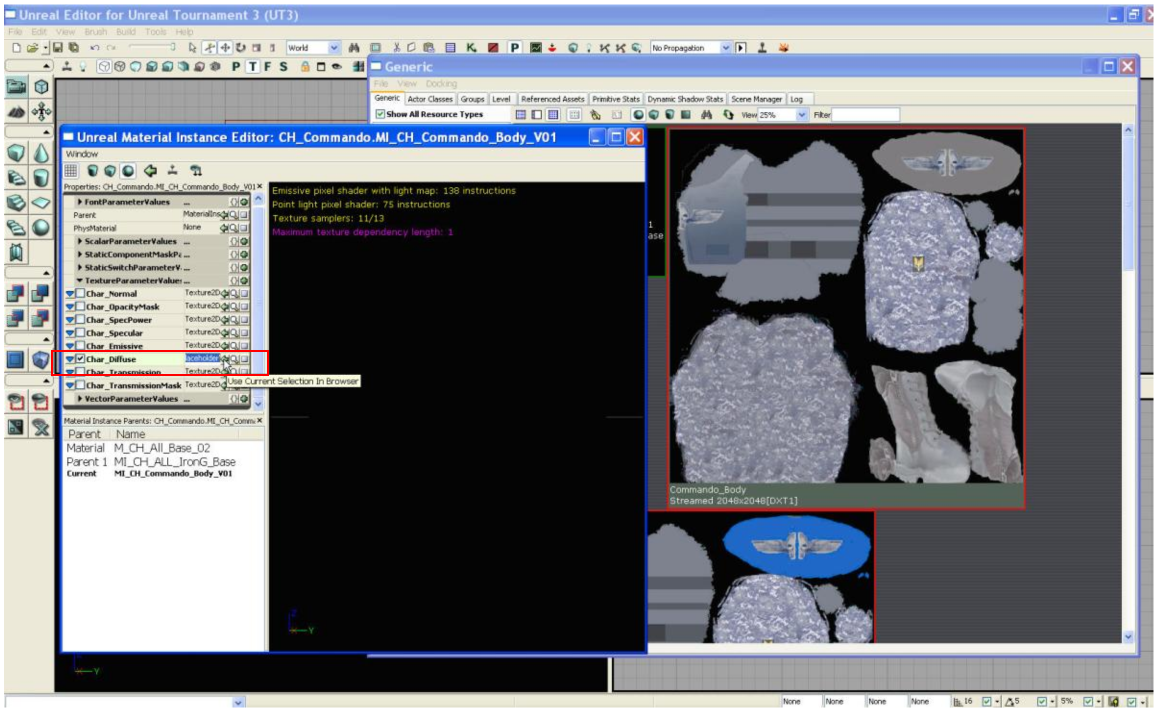
```
Emissive pixel shader with light map: 138 instructions
Point light pixel shader: 75 instructions
Texture samplers: 11/13
Maximum texture dependency length: 1
```

Material Instance Parents: CH_Commando.MI_CH_Comm

Parent	Name
Material	M_CH_All_Base_02
Parent 1	MI_CH_ALL_IronG_Base
Current	MI_CH_Commando_Body_V01

```
Commando_Body
Streamed 2048x2048[DXT1]
```

Then in the Material instance editor window, for our body MIC as shown below, go to the Texture parameter values tab on the left. And expand it.


Unreal Editor for Unreal Tournament 3 (UT3)
Generic
Unreal Material Instance Editor: CH_Commando.MI_CH_Commando_Body_V01
Use Current Selection In Browser
Commando_Body
Streamed 2048x2048[DXT1]

Properties: CH_Commando.MI_CH_Commando_Body_V01

FontParameterValues	...
Parent	MaterialIns
PhysMaterial	None
ScalarParameterValues	...
StaticComponentMaskP	...
StaticSwitchParameterV	...
TextureParameterValue	...
Char_Normal	Texture2D
Char_OpacityMask	Texture2D
Char_SpecPower	Texture2D
Char_Specular	Texture2D
Char_Emissive	Texture2D
[✓] Char_Diffuse	aceholder
Char_Transmission	Texture2D
Char_TransmissionMask	Texture2D
VectorParameterValues	...

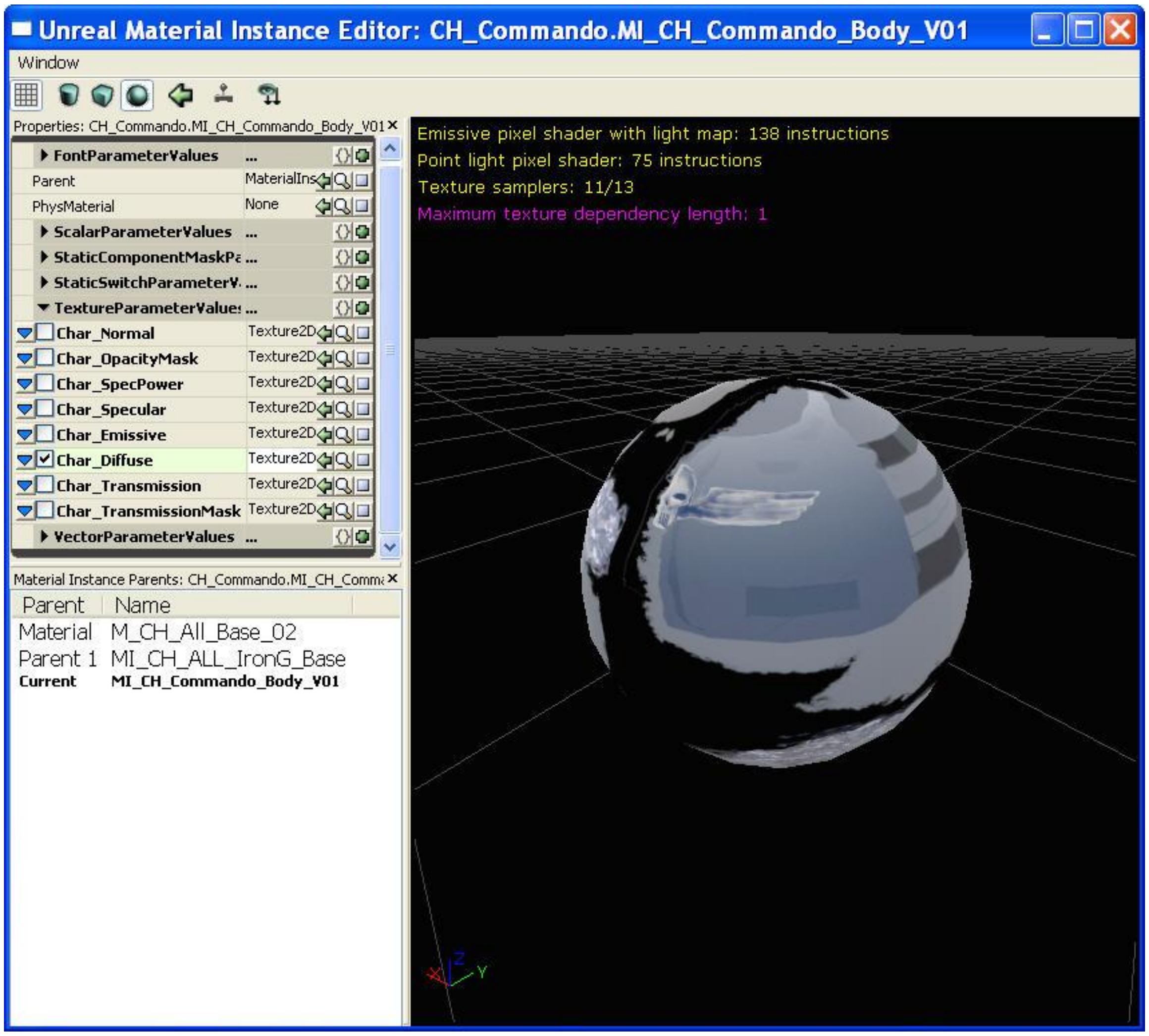
```
Emissive pixel shader with light map: 138 instructions
Point light pixel shader: 75 instructions
Texture samplers: 11/13
Maximum texture dependency length: 1
```

Material Instance Parents: CH_Commando.MI_CH_Comm

Parent	Name
Material	M_CH_All_Base_02
Parent 1	MI_CH_ALL_IronG_Base
Current	MI_CH_Commando_Body_V01

There are now numerous channels displayed with joining tick-boxes. Select the diffuse check box. Then select the “use selection from browser” button, the small green arrow as shown in the boxed area in the above picture.

This will add our Body diffuse texture to our Body Material as shown on the picture below.


Unreal Material Instance Editor: CH_Commando.MI_CH_Commando_Body_V01
Window
Properties: CH_Commando.MI_CH_Commando_Body_V01

FontParameterValues	...
Parent	MaterialIns
PhysMaterial	None
ScalarParameterValues	...
StaticComponentMaskPa	...
StaticSwitchParameterV	...
TextureParameterValues	...
Char_Normal	Texture2D
Char_OpacityMask	Texture2D
Char_SpecPower	Texture2D
Char_Specular	Texture2D
Char_Emissive	Texture2D
[✓] Char_Diffuse	Texture2D
Char_Transmission	Texture2D
Char_TransmissionMask	Texture2D
VectorParameterValues	...

```
Emissive pixel shader with light map: 138 instructions
Point light pixel shader: 75 instructions
Texture samplers: 11/13
Maximum texture dependency length: 1
```

Material Instance Parents: CH_Commando.MI_CH_Comm

Parent	Name
Material	M_CH_All_Base_02
Parent 1	MI_CH_ALL_IronG_Base
Current	MI_CH_Commando_Body_V01

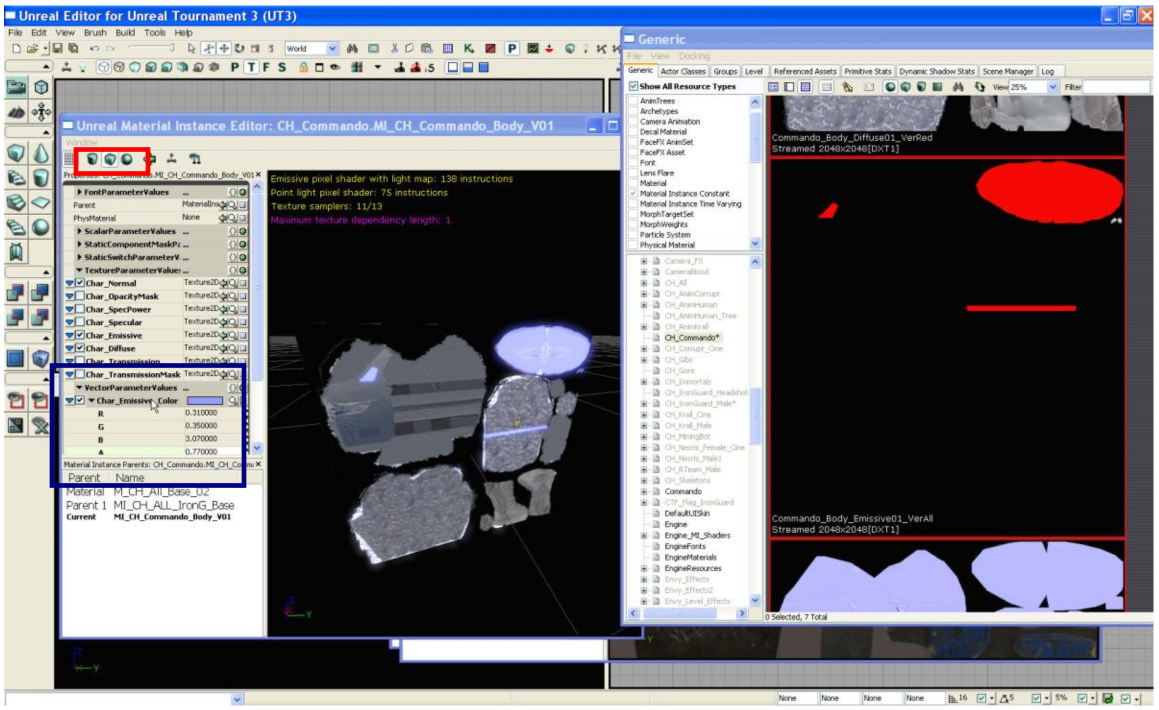
Illustration above shows the Commando_Body material with the diffuse texture applied to it.

Now this process has to be repeated for all of the textures that were imported.

So the Normal Map should be connected to the Char_Normal attribute. The specular map for the Char_specular attribute and so on.

You can view the texture/material differently by using the swatch type tabs at the top of the editor.(shown in the red box, in the picture below.)

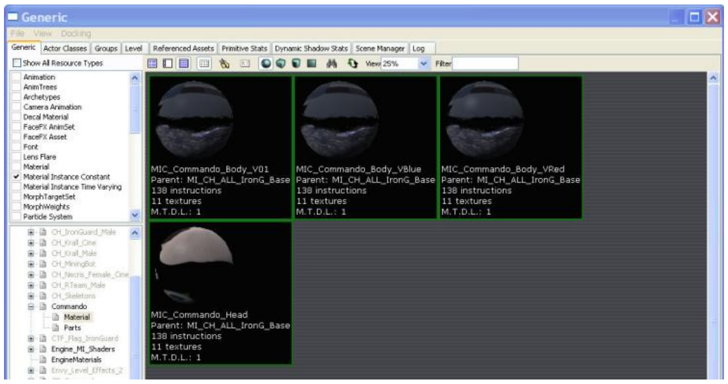
If you have an “emissive” texture. Connect it to the Char_emissive channel. The next image shows how to get it to work correctly. An Emissive texture is just a black and white solid colour image that depicts areas on the material that “glow” or have an emissive light and colour. So it’s basically painting a small light onto your model.


Unreal Editor for Unreal Tournament 3 (UT3)
Unreal Material Instance Editor: CH_Commando.MI_CH_Commando_Body_V01
Generic
VectorParameterValues
Char_Emissive_Color

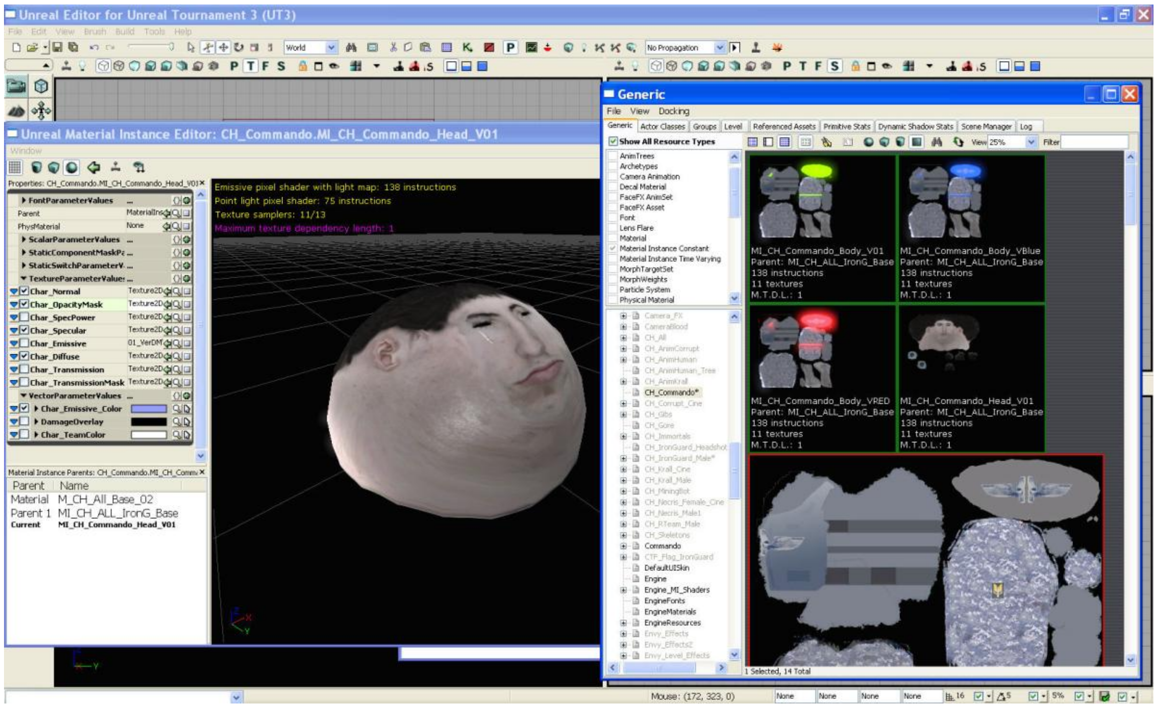
This illustration shows an example emissive texture. At first the material will glow a very solid bright white colour. To edit this expand the VectorParameterValues tab in the Material Instance editor. (Depicted in the blue box above)
Then select the Char_emissive_Color channel check box and expand the tab. This will provide Red, green, Blue and alpha values for the emission. Reducing the alpha will make the glow less intense and the diffuse texture underneath more visible. Changing the Colour values will change the emission colour, as in the picture above.

This should finish the work on our first material for the body. However we still need materials for a blue version, a red version and a material for the head.

Rather than re-creating more MIC’s from scratch the one we just made can be duplicated and renamed by right clicking on our material swatch in the generic browser, and selecting duplicate. Then right click on the copy and select rename. Rename this new MIC to “MIC_[packageName]_Body_VBlue” then duplicate it again and rename to “MIC_[packageName]_Body_VRed” then one last time and rename it: “MIC_[packageName]_Head”
As shown in the picture below.



Now swap the diffuse textures for the blue and red versions. Then if you haven't already import the head textures that were created earlier. Then connect them up into the correct channels in the material instance editor for the Head MIC. So you end up with the picture below.


Unreal Editor for Unreal Tournament 3 (UT3)
Unreal Material Instance Editor: CH_Commando.MI_CH_Commando_Head_V01
Generic

Properties: CH_Commando.MI_CH_Commando_Head_V01

FontParameterValues	...	
Parent	MaterialIns	
PhysMaterial	None	
ScalarParameterValues	...	
StaticComponentMaskPa	...	
StaticSwitchParameterV	...	
TextureParameterValues	...	
☑ Char_Normal	Texture2D	
☑ Char_OpacityMask	Texture2D	
☐ Char_SpecPower	Texture2D	
☑ Char_Specular	Texture2D	
☐ Char_Emissive	01_VerDM	
☑ Char_Diffuse	Texture2D	
☐ Char_Transmission	Texture2D	
☐ Char_TransmissionMask	Texture2D	
VectorParameterValues	...	
☑ Char_Emissive_Color	🟦	
☐ DamageOverlay	⬛	
☐ Char_TeamColor	⬜	

Emissive pixel shader with light map: 138 instructions
Point light pixel shader: 75 instructions
Texture samplers: 11/13
Maximum texture dependency length: 1

Material Instance Parents: CH_Commando.MI_CH_Comm

Parent	Name
Material	M_CH_All_Base_02
Parent 1	MI_CH_ALL_IronG_Base
Current	MI_CH_Commando_Head_V01

MI_CH_Commando_Body_V01
Parent: MI_CH_ALL_IronG_Base
138 instructions
11 textures
M.T.D.L.: 1

MI_CH_Commando_Body_VBlue
Parent: MI_CH_ALL_IronG_Base
138 instructions
11 textures
M.T.D.L.: 1

MI_CH_Commando_Body_VRED
Parent: MI_CH_ALL_IronG_Base
138 instructions
11 textures
M.T.D.L.: 1

MI_CH_Commando_Head_V01
Parent: MI_CH_ALL_IronG_Base
138 instructions
11 textures
M.T.D.L.: 1

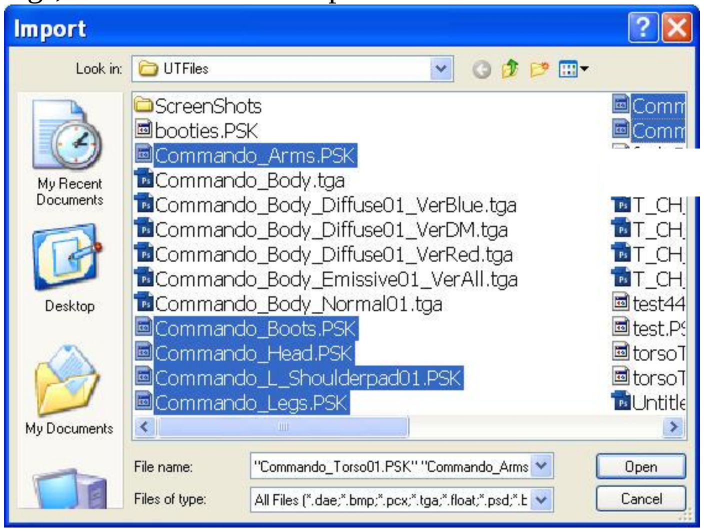
1 Selected, 14 Total

The Char_TeamColor Channel should be enabled for the team colour materials to work correctly.

This should conclude the work needed on materials.

Importing the Model/mesh

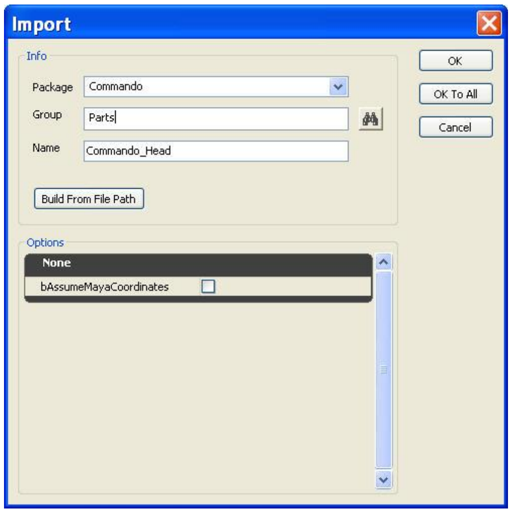
In the generic browser window go to file -> import Select all of the meshes, Arms, head, legs, torso etc. As in the picture below.



ScreenShots
booties.PSK
Commando_Arms.PSK
Commando_Body.tga
Commando_Body_Diffuse01_VerBlue.tga
Commando_Body_Diffuse01_VerDM.tga
Commando_Body_Diffuse01_VerRed.tga
Commando_Body_Emissive01_VerAll.tga
Commando_Body_Normal01.tga
Commando_Boots.PSK
Commando_Head.PSK
Commando_L_Shoulderpad01.PSK
Commando_Legs.PSK

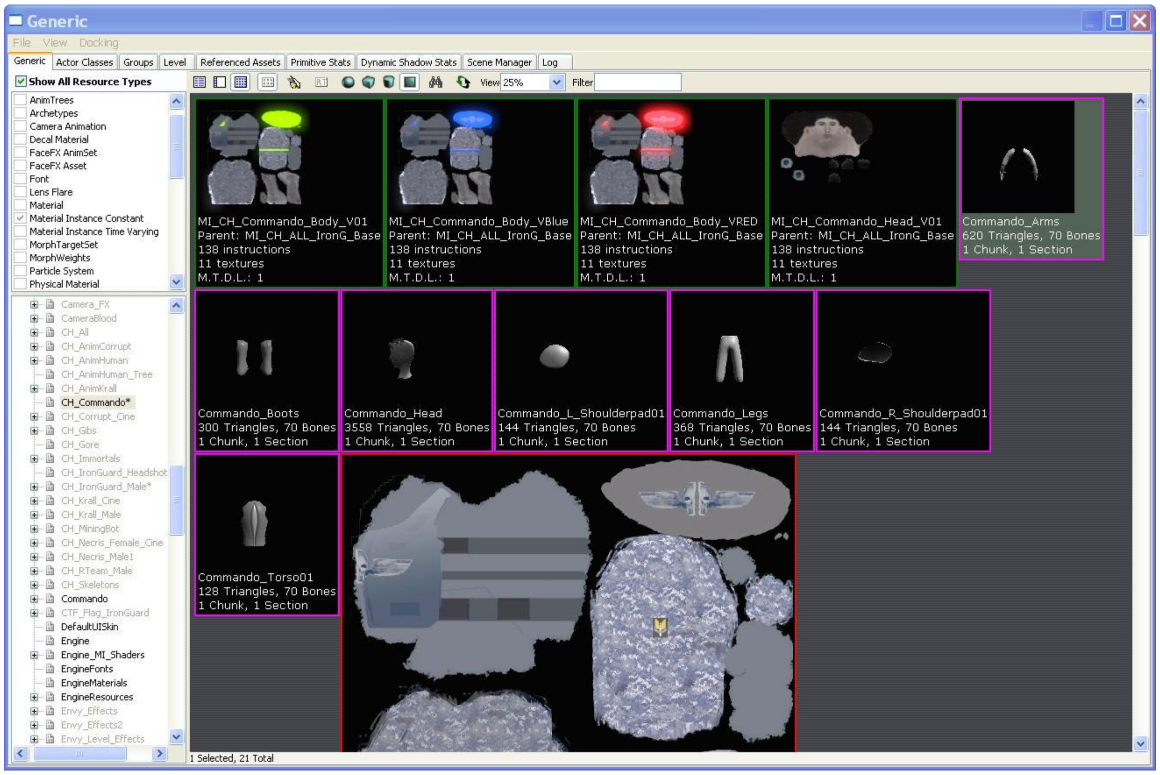
File name: "Commando_Torso01.PSK" "Commando_Arms
Files of type: All Files (*.dae;*.bmp;*.pcx;*.tga;*.float;*.psd;*.t

When the import dialogue box appears make sure that the Package name is correct and the group should be labelled "**Parts**" as shown below. Then click ok to all to import all of the mesh parts.



Package: Commando
Group: Parts
Name: Commando_Head
bAssumeMayaCoordinates ☐

*note in "unreal ed" there is no need to have the "assume Maya cords" box checked the "actorX" plug-in already accounts for this.



MI_CH_Commando_Body_V01
Parent: MI_CH_ALL_IronG_Base
138 instructions
11 textures
M.T.D.L.: 1

MI_CH_Commando_Body_VBlue
Parent: MI_CH_ALL_IronG_Base
138 instructions
11 textures
M.T.D.L.: 1

MI_CH_Commando_Body_VRED
Parent: MI_CH_ALL_IronG_Base
138 instructions
11 textures
M.T.D.L.: 1

MI_CH_Commando_Head_V01
Parent: MI_CH_ALL_IronG_Base
138 instructions
11 textures
M.T.D.L.: 1

Commando_Arms
620 Triangles, 70 Bones
1 Chunk, 1 Section

Commando_Boots
300 Triangles, 70 Bones
1 Chunk, 1 Section

Commando_Head
3558 Triangles, 70 Bones
1 Chunk, 1 Section

Commando_L_Shoulderpad01
144 Triangles, 70 Bones
1 Chunk, 1 Section

Commando_Legs
368 Triangles, 70 Bones
1 Chunk, 1 Section

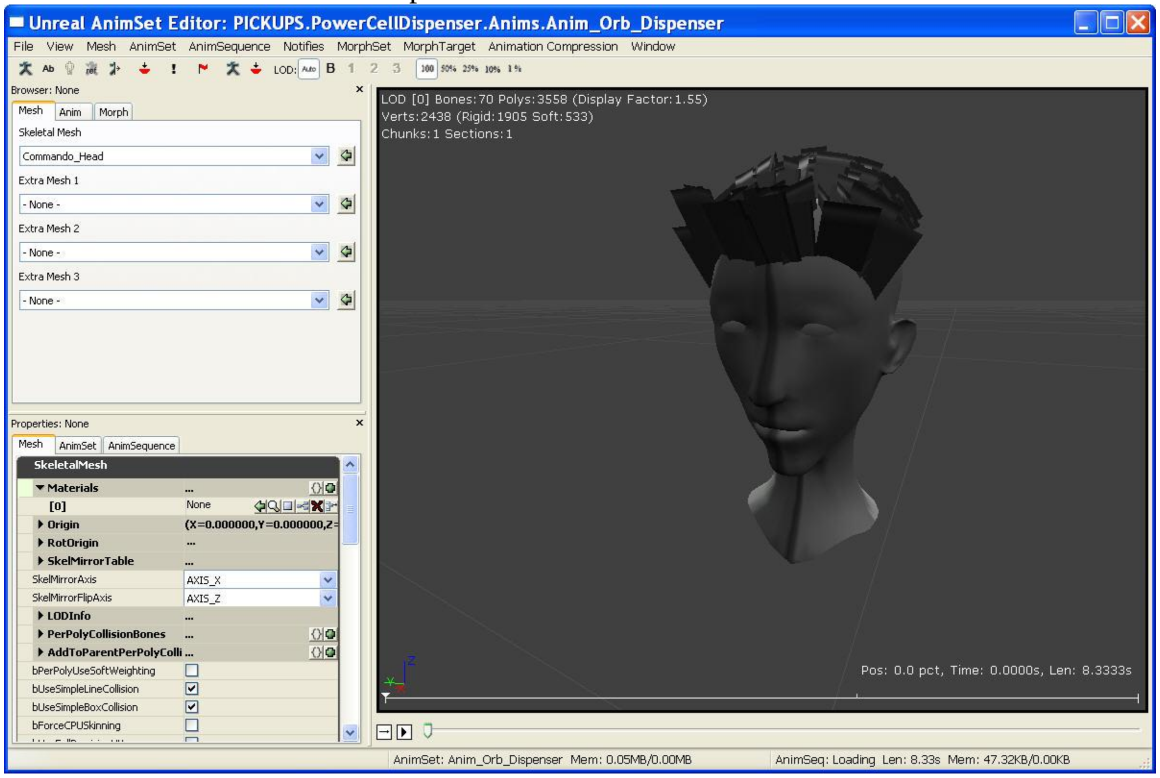
Commando_R_Shoulderpad01
144 Triangles, 70 Bones
1 Chunk, 1 Section

Commando_Torso01
128 Triangles, 70 Bones
1 Chunk, 1 Section

1 Selected, 21 Total

The picture above shows the imported parts in the generic browser with the default checker material applied. As shown I used the most polys in the head mesh because I thought it to be the most important detail, to be able to recognise.

Double click on the “head” to open the Anim set editor.



Unreal AnimSet Editor: PICKUPS.PowerCellDispenser.Anims.Anim_Orb_Dispenser
File View Mesh AnimSet AnimSequence Notifies MorphSet MorphTarget Animation Compression Window
Browser: None
Mesh Anim Morph
Properties: None
Mesh AnimSet AnimSequence

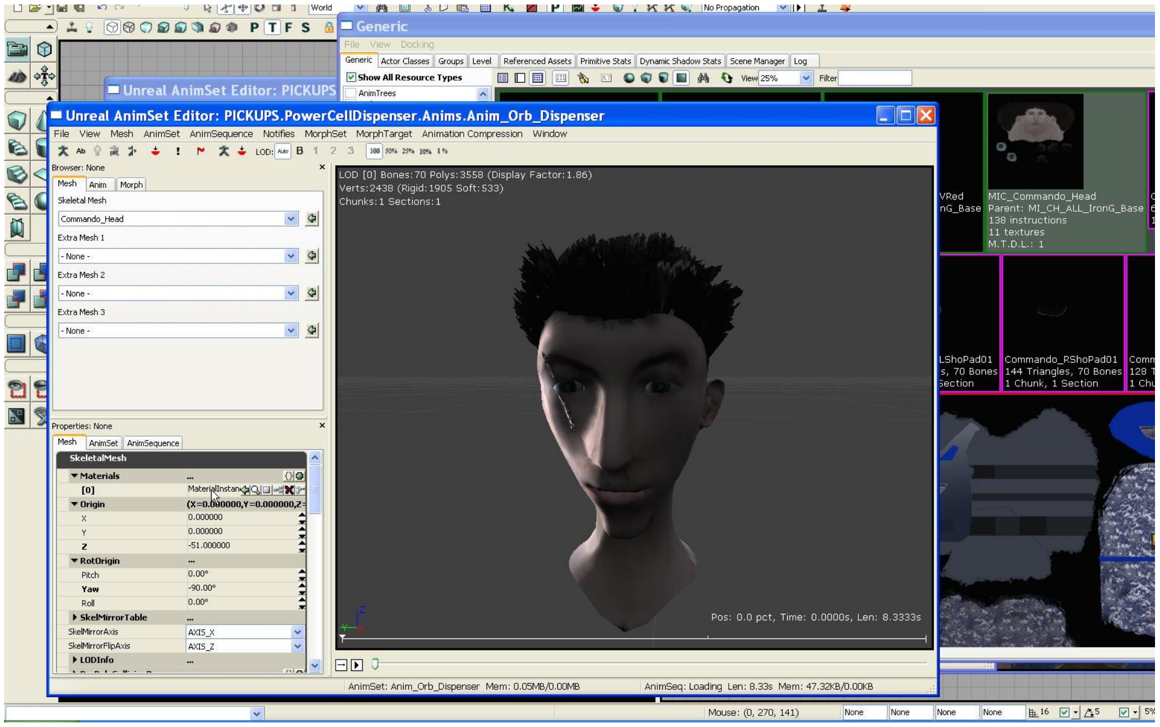
Skeletal Mesh: Commando_Head
Extra Mesh 1: - None -
Extra Mesh 2: - None -
Extra Mesh 3: - None -

LOD [0] Bones:70 Polys:3558 (Display Factor:1.55)
Verts:2438 (Rigid:1905 Soft:533)
Chunks:1 Sections:1

Pos: 0.0 pct, Time: 0.0000s, Len: 8.3333s

AnimSet: Anim_Orb_Dispenser Mem: 0.05MB/0.00MB
AnimSeq: Loading Len: 8.33s Mem: 47.32KB/0.00KB

Under the mesh tab open the materials tab and connect the Head MIC material into the first channel marked [0] as shown in the picture below.



Generic
Unreal AnimSet Editor: PICKUPS.PowerCellDispenser.Anims.Anim_Orb_Dispenser
Commando_RShoPad01
144 Triangles, 70 Bones
1 Chunk, 1 Section

MIC_Commando_Head
Parent: MI_CH_ALL_IronG_Base
138 instructions
11 textures
M.T.D.L.: 1

LOD [0] Bones:70 Polys:3558 (Display Factor:1.86)
Verts:2438 (Rigid:1905 Soft:533)
Chunks:1 Sections:1

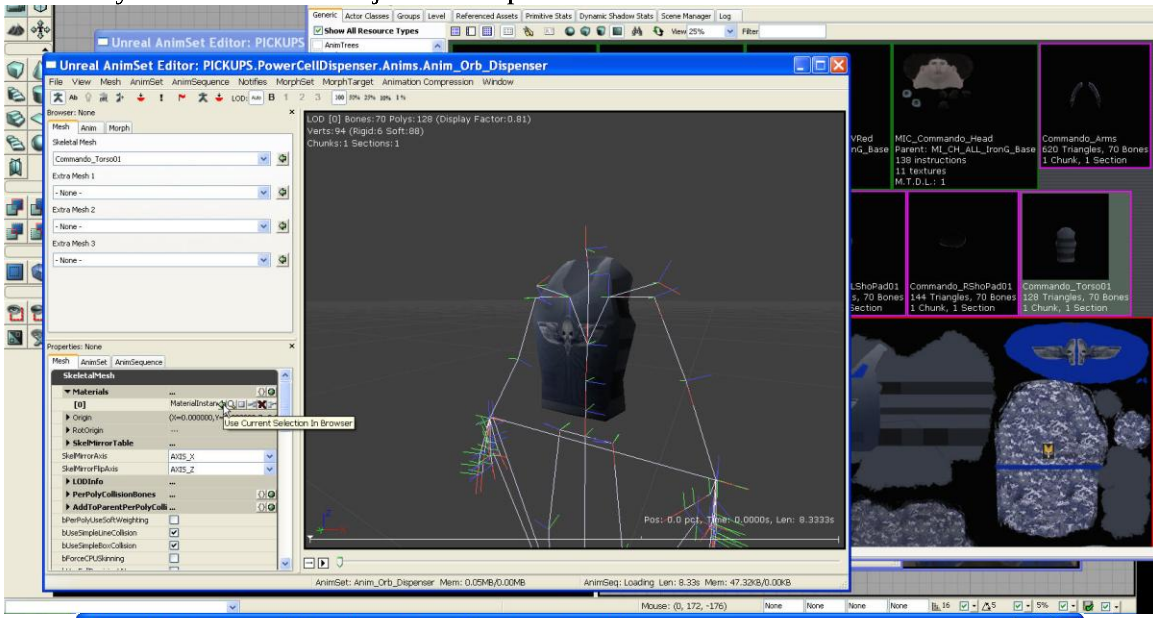
Pos: 0.0 pct, Time: 0.0000s, Len: 8.3333s

AnimSet: Anim_Orb_Dispenser Mem: 0.05MB/0.00MB
AnimSeq: Loading Len: 8.33s Mem: 47.32KB/0.00KB

Mouse: (0, 270, 141)

For the head mesh only for the game to calculate rotations correctly the head object's origin and rotation origin needs to be edited as in the picture above. The Z coordinate has to be set to -51 and the Yaw has to be set to -90 degrees. This is for the head object **only.**

Then all of the other parts need to have the body MIC material applied to them in the same way. Like the torso object in the picture below.


Unreal AnimSet Editor: PICKUPS.PowerCellDispenser.Anims.Anim_Orb_Dispenser

Skeletal Mesh: Commando_Torso01

LOD [0] Bones:70 Polys:128 (Display Factor:0.81)
Verts:94 (Rigid:6 Soft:88)
Chunks:1 Sections:1

Pos: 0.0 pct, Time: 0.0000s, Len: 8.3333s

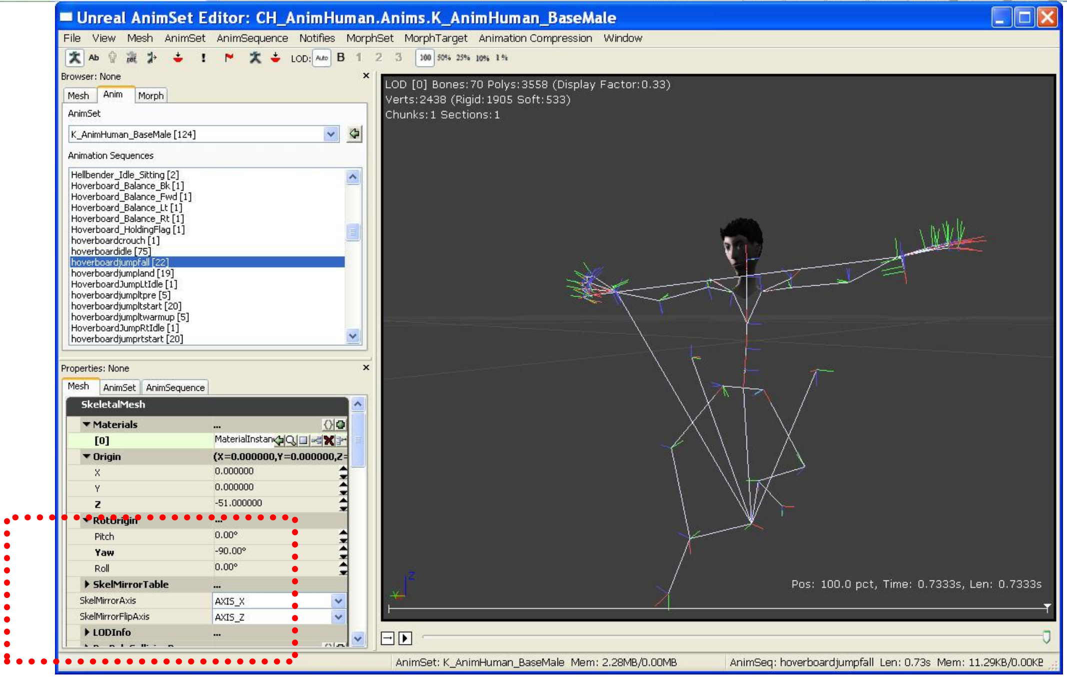
AnimSet: Anim_Orb_Dispenser Mem: 0.05MB/0.00MB   AnimSeq: Loading Len: 8.33s Mem: 47.32KB/0.00KB

MIC_Commando_Head
Parent: MI_CH_ALL_IronG_Base
138 instructions
11 textures
M.T.D.L.: 1

Commando_Arms
620 Triangles, 70 Bones
1 Chunk, 1 Section

Commando_RShoPad01
144 Triangles, 70 Bones
1 Chunk, 1 Section

Commando_Torso01
128 Triangles, 70 Bones
1 Chunk, 1 Section


Unreal AnimSet Editor: CH_AnimHuman.Anims.K_AnimHuman_BaseMale

AnimSet: K_AnimHuman_BaseMale [124]

Animation Sequences:
Hellbender_Idle_Sitting [2]
Hoverboard_Balance_Bk [1]
Hoverboard_Balance_Fwd [1]
Hoverboard_Balance_Lt [1]
Hoverboard_Balance_Rt [1]
Hoverboard_HoldingFlag [1]
hoverboardcrouch [1]
hoverboardidle [75]
hoverboardjumpfall [22]
hoverboardjumpland [19]
HoverboardJumpLtIdle [1]
hoverboardjumpltpre [5]
hoverboardjumpltstart [20]
hoverboardjumpltwarmup [5]
HoverboardJumpRtIdle [1]
hoverboardjumprtstart [20]

Materials	...
[0]	MaterialInstanc
Origin	(X=0.000000,Y=0.000000,Z=
X	0.000000
Y	0.000000
Z	-51.000000
RotOrigin	...
Pitch	0.00°
Yaw	-90.00°
Roll	0.00°
SkelMirrorTable	...
SkelMirrorAxis	AXIS_X
SkelMirrorFlipAxis	AXIS_Z
LODInfo	...

LOD [0] Bones:70 Polys:3558 (Display Factor:0.33)
Verts:2438 (Rigid:1905 Soft:533)
Chunks:1 Sections:1

Pos: 100.0 pct, Time: 0.7333s, Len: 0.7333s

AnimSet: K_AnimHuman_BaseMale Mem: 2.28MB/0.00MB   AnimSeq: hoverboardjumpfall Len: 0.73s Mem: 11.29KB/0.00KB

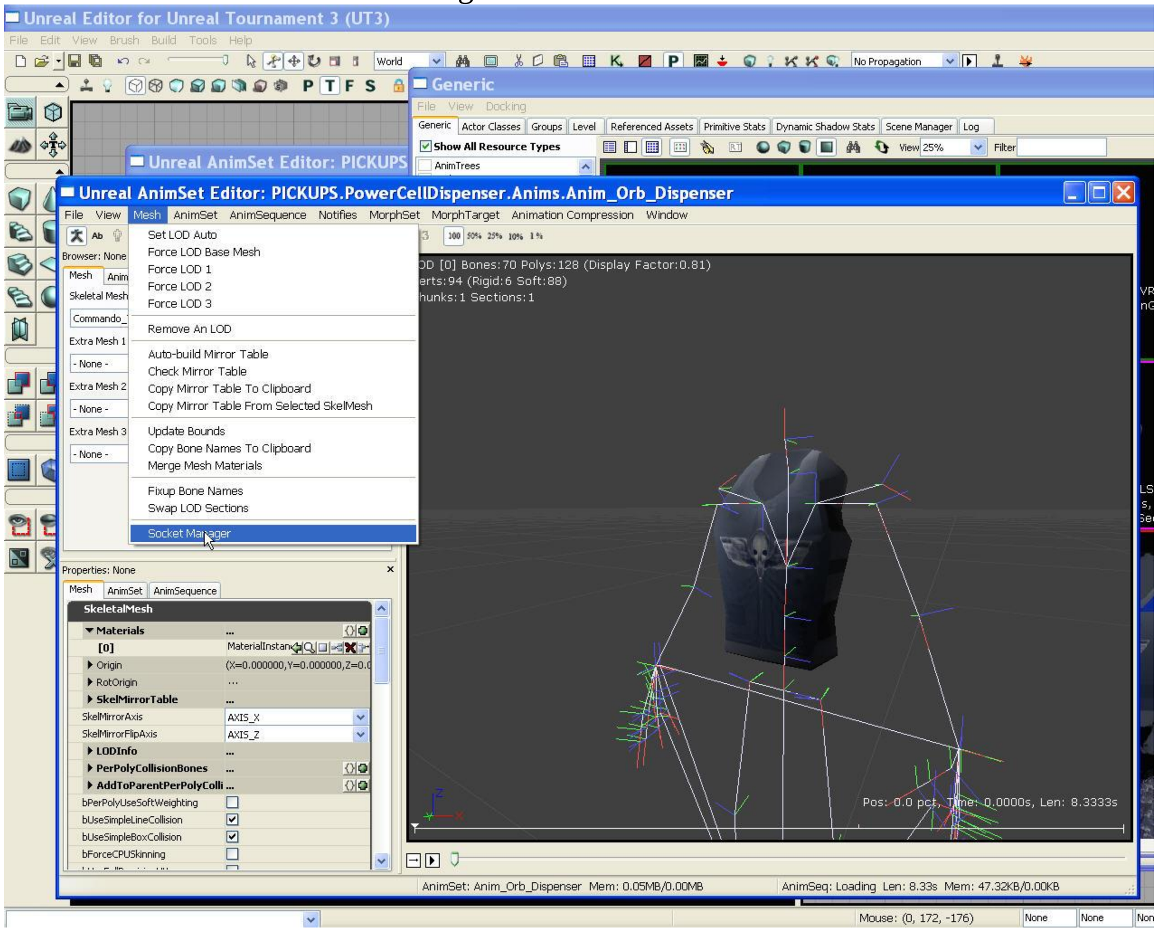
Once the objects have texture and the head object has been edited it possible to view the animations by selecting the Anim tab, in the anim set editor. Then in the anim set drop down box select K_AnimHuman_BaseMale[124]. Then chose any of the pre-made animations. If the skeleton I provided was used and the previous instructions were followed correctly, there shouldn't be any problems with the animation playback. As shown in the picture above.

Sockets

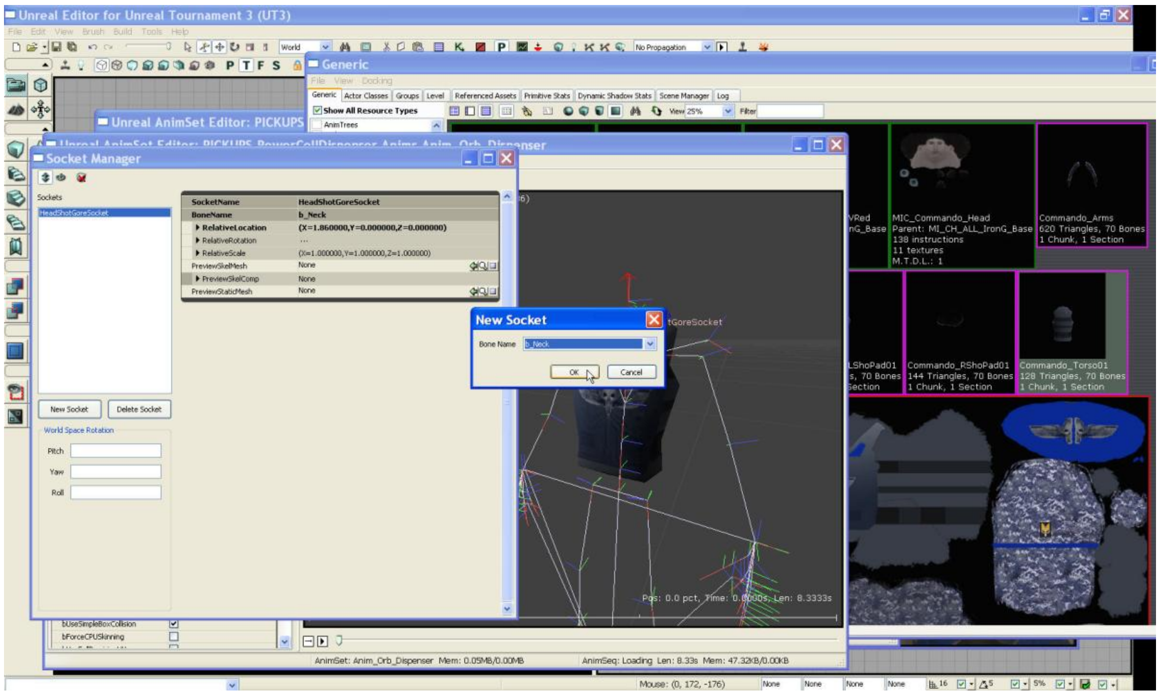
Sockets are basically locators that are affixed to geometry to inform the game engine of certain interactive procedures. For example even thought the character has hands the game engine doesn’t know where or how the character holds a weapon. So Sockets need to be placed onto some of the parts, so that the character will work and interact with the game environment properly.

The parts that need sockets are the torso, boots/feet and arms.

Starting with the torso add a socket by entering the socket manager from the anim set editor ->mesh menu-> socket manager. As shown below.

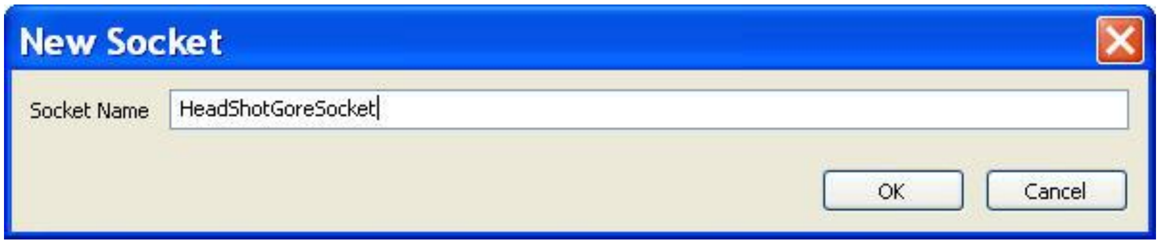

Unreal Editor for Unreal Tournament 3 (UT3)
Unreal AnimSet Editor: PICKUPS.PowerCellDispenser.Anims.Anim_Orb_Dispenser
File View Mesh AnimSet AnimSequence Notifies MorphSet MorphTarget Animation Compression Window
Set LOD Auto
Force LOD Base Mesh
Force LOD 1
Force LOD 2
Force LOD 3
Remove An LOD
Auto-build Mirror Table
Check Mirror Table
Copy Mirror Table To Clipboard
Copy Mirror Table From Selected SkelMesh
Update Bounds
Copy Bone Names To Clipboard
Merge Mesh Materials
Fixup Bone Names
Swap LOD Sections
Socket Manager
Generic
Properties: None
SkeletalMesh
Materials
Origin
RotOrigin
SkelMirrorTable
SkelMirrorAxis AXIS_X
SkelMirrorFlipAxis AXIS_Z
LODInfo
PerPolyCollisionBones
AddToParentPerPolyColli
bPerPolyUseSoftWeighting
bUseSimpleLineCollision
bUseSimpleBoxCollision
bForceCPUSkinning
Pos: 0.0 pct, Time: 0.0000s, Len: 8.3333s
AnimSet: Anim_Orb_Dispenser Mem: 0.05MB/0.00MB
AnimSeq: Loading Len: 8.33s Mem: 47.32KB/0.00KB

Inside the socket manager select add new socket. A dialogue box will appear with a list of the bones in our characters skeleton. Select b_neck. As shown below



SocketName	HeadShotGoreSocket
BoneName	b_Neck
RelativeLocation	(X=1.860000,Y=0.000000,Z=0.000000)
RelativeRotation	...
RelativeScale	(X=1.000000,Y=1.000000,Z=1.000000)
PreviewSkelMesh	None
PreviewSkelComp	None
PreviewStaticMesh	None

Then it will ask for a Socket name Enter: HeadShotGoreSocket.
This socket is fixed to the top of the neck, so if the player is hit there it will register a headshot and kill the player. Etc... *Note it could be counted as cheating if you move the headshot socket somewhere else say 50 feet away from your character…

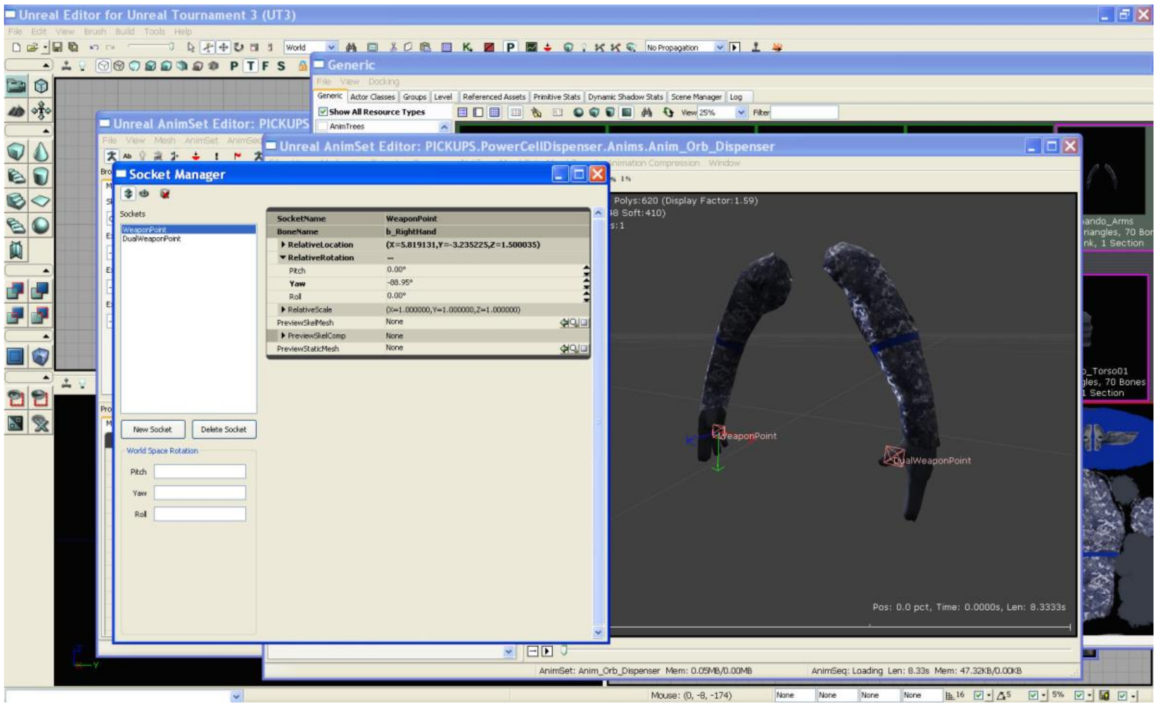


Socket Name: HeadShotGoreSocket

Using the on screen controls position the socket in a sensible place at the base of the neck/head

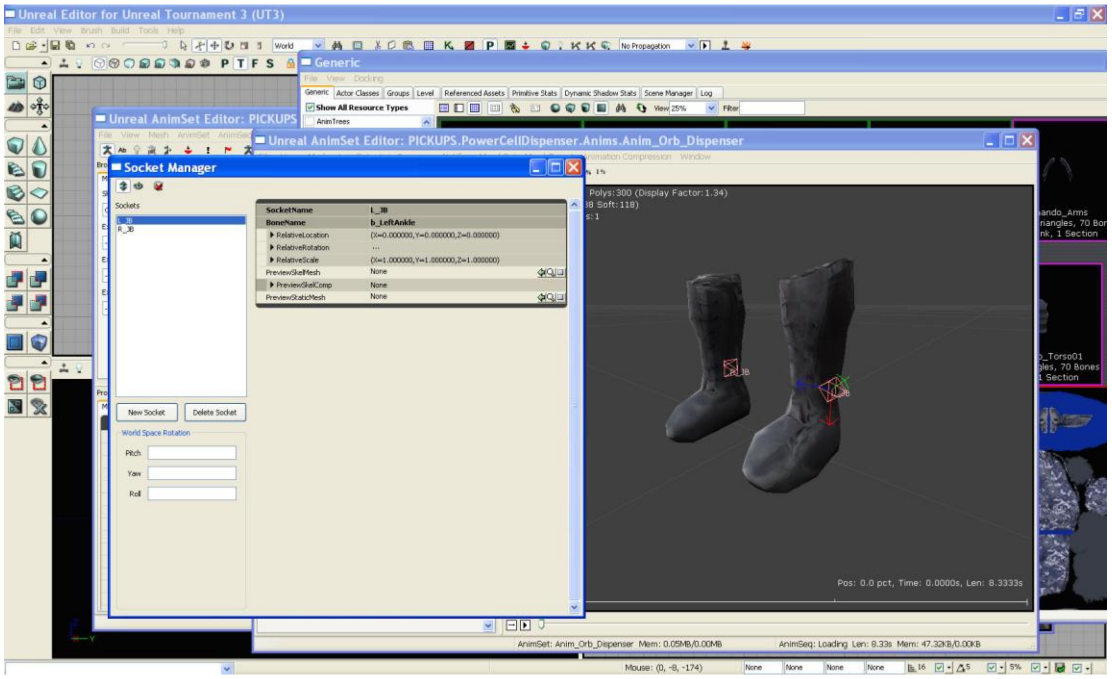
Now select ok and close the anim set viewer.

Now for the arms, open the anim set viewer for the arms and the socket manager. The arms need two sockets. One for each weapon hand. As shown below


Unreal Editor for Unreal Tournament 3 (UT3)
Generic
Unreal AnimSet Editor: PICKUPS
Unreal AnimSet Editor: PICKUPS.PowerCellDispenser.Anims.Anim_Orb_Dispenser
Socket Manager
Sockets
WeaponPoint
DualWeaponPoint
New Socket
Delete Socket
World Space Rotation
Pitch
Yaw
Roll
WeaponPoint
DualWeaponPoint
Polys:620 (Display Factor:1.59)
Pos: 0.0 pct, Time: 0.0000s, Len: 8.3333s

SocketName	WeaponPoint
BoneName	b_RightHand
RelativeLocation	(X=5.819131,Y=-3.235225,Z=1.500035)
RelativeRotation	...
Pitch	0.00°
Yaw	-88.95°
Roll	0.00°
RelativeScale	(X=1.000000,Y=1.000000,Z=1.000000)
PreviewSkelMesh	None
PreviewSkelComp	None
PreviewStaticMesh	None

Add a new socket affix he first to the b_rightHand joint, name the socket WeaponPoint. If your character is right-handed. Then add the second socket and affix it to the b_LeftHand joint. Name this socket DualWeaponPoint. The sockets will then have to be moved visually to line up properly. Move and rotate the sockets to look like the image above. X- Axis pointing in, z pointing out, Y pointing up. This step took me hours to get the positioning just right. May have to come back to this step later after seeing the model in unreal Tournament.

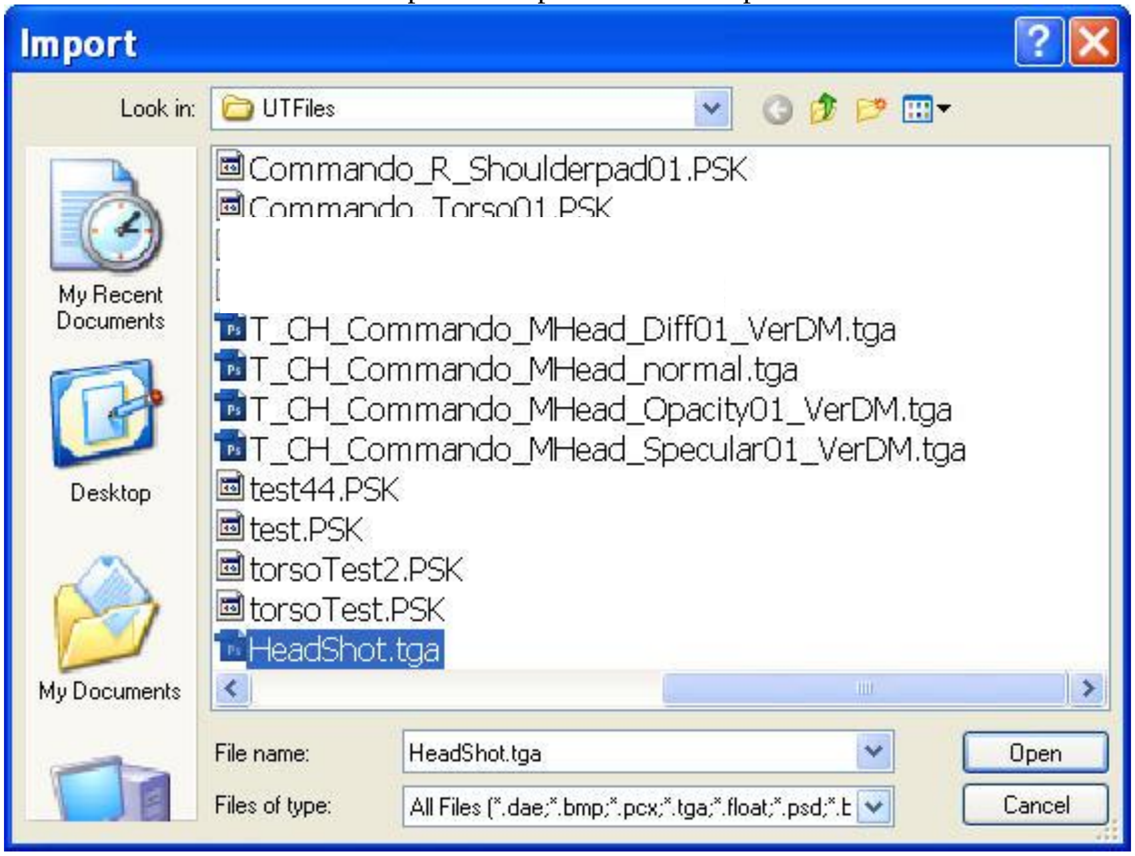

Unreal Editor for Unreal Tournament 3 (UT3)
Generic
Unreal AnimSet Editor: PICKUPS
Unreal AnimSet Editor: PICKUPS.PowerCellDispenser.Anims.Anim_Orb_Dispenser
Socket Manager
Sockets
L_JB
R_JB
New Socket
Delete Socket
World Space Rotation
Pitch
Yaw
Roll
Polys:300 (Display Factor:1.34)
Pos: 0.0 pct, Time: 0.0000s, Len: 8.3333s

SocketName	L_JB
BoneName	b_LeftAnkle
RelativeLocation	(X=0.000000,Y=0.000000,Z=0.000000)
RelativeRotation	...
RelativeScale	(X=1.000000,Y=1.000000,Z=1.000000)
PreviewSkelMesh	None
PreviewSkelComp	None
PreviewStaticMesh	None

Now the boots need two Sockets. One named L_JB, affix that to the b_LeftAnkle joint. The second Named R_JB and affix that to the right ankle joint.

Portrait Image

To be able to display a portrait picture, need to import one. A 256x 256 will suffice. So from the file menu select import and open a sufficient picture.



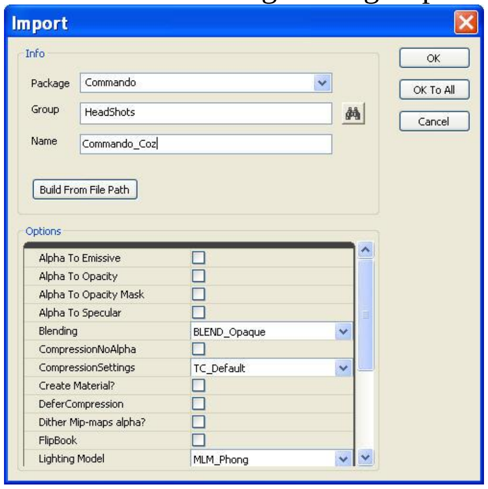
Look in: UTFiles

Commando_R_Shoulderpad01.PSK
Commando_Torso01.PSK
T_CH_Commando_MHead_Diff01_VerDM.tga
T_CH_Commando_MHead_normal.tga
T_CH_Commando_MHead_Opacity01_VerDM.tga
T_CH_Commando_MHead_Specular01_VerDM.tga
test44.PSK
test.PSK
torsoTest2.PSK
torsoTest.PSK
HeadShot.tga

File name: HeadShot.tga

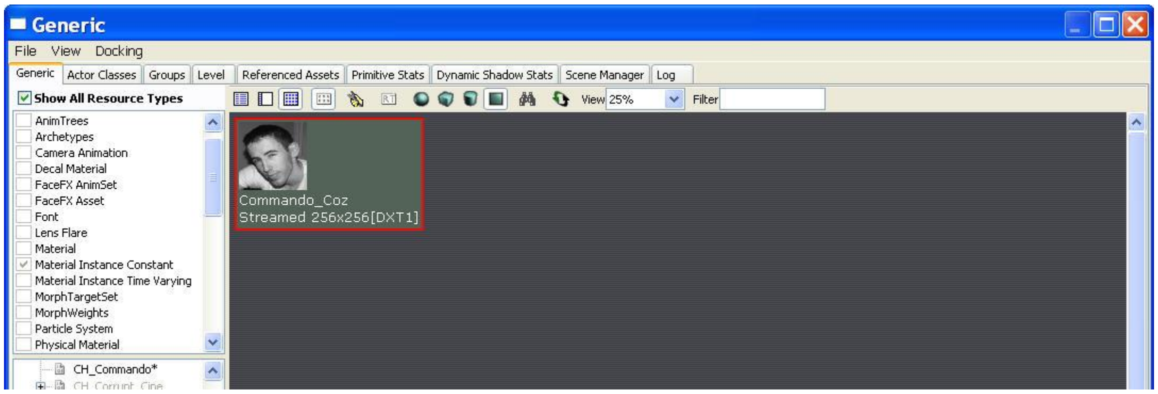
Files of type: All Files (*.dae;*.bmp;*.pcx;*.tga;*.float;*.psd;*.b

The head shot settings. The group should be "HeadShots", the name [name]

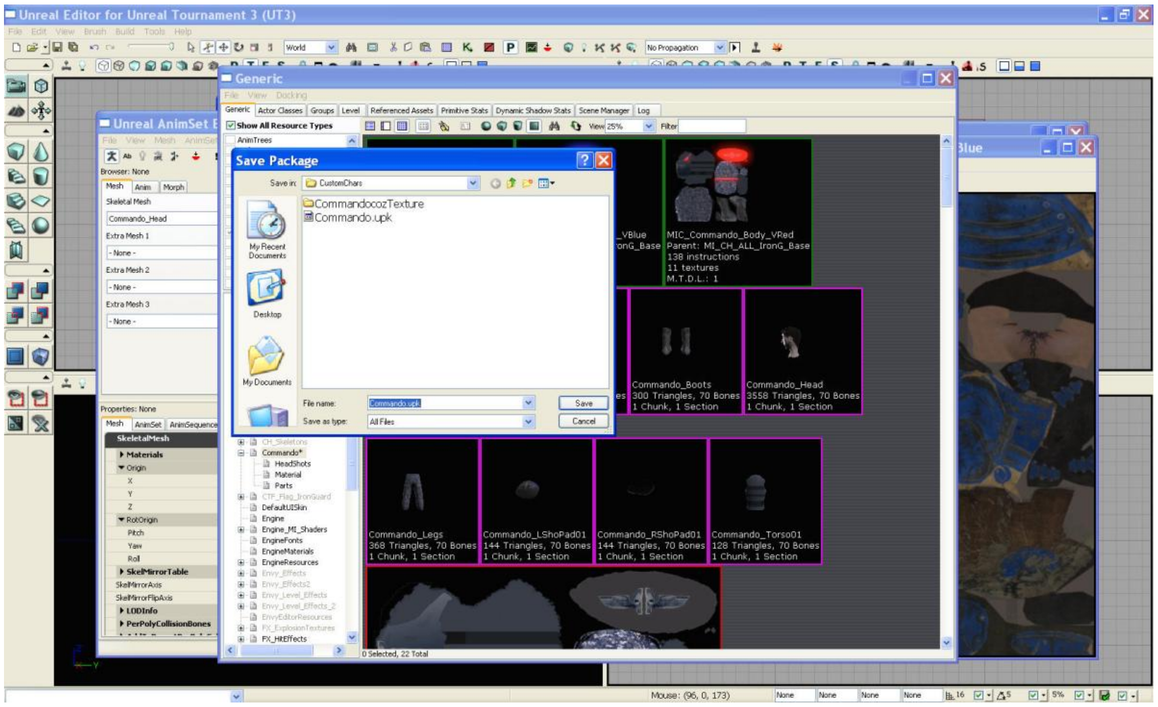


Package: Commando
Group: HeadShots
Name: Commando_Coz

Alpha To Emissive	☐
Alpha To Opacity	☐
Alpha To Opacity Mask	☐
Alpha To Specular	☐
Blending	BLEND_Opaque
CompressionNoAlpha	☐
CompressionSettings	TC_Default
Create Material?	☐
DeferCompression	☐
Dither Mip-maps alpha?	☐
FlipBook	☐
Lighting Model	MLM_Phong



Example headshot



Last thing is to save the package. To the folder below. Also depicted in the illustration above.

C:\Documents and Settings\[User]\My Documents\My Games\Unreal Tournament 3\UTGame\Published\CookedPC\CustomChars

This only leaves the scripting component to do. So the game will “know” that it has a new character available to use.

Scripting Component

The code in the appendix starting with the following line:

Parts=(Part=PART_Head,ObjectName="Commando.Parts.Commando_Head",PartID="0A",FamilyID="IRNM")

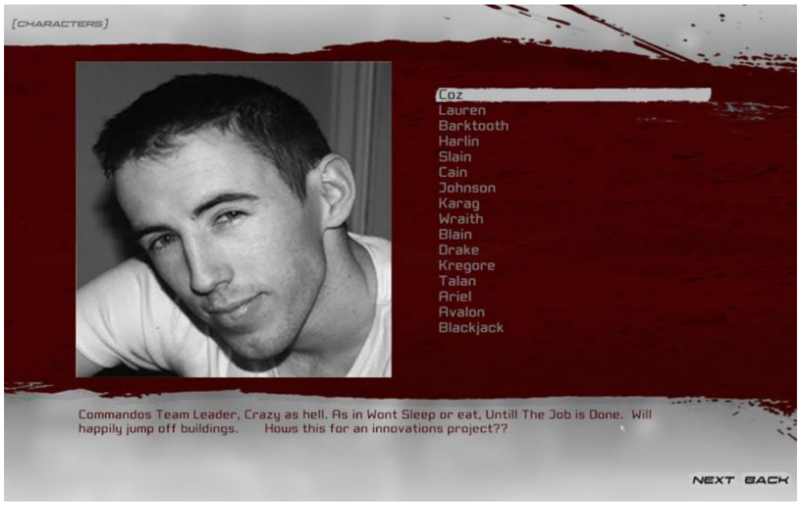
Needs to be appended to the default UTCustomChar.ini which can be found in the folder:
C:\Documents and Settings\[user]\My Documents\My Games\Unreal Tournament 3\UTGame\Config

The “Parts Head” bit speaks for itself, the Object name consists of [packageName].[GroupName].[PartName]

PartID is to identify parts within a familyID. The FamilyID is the Faction ID Number for example iron Guard or TwinSouls.

In Game ScreenShots

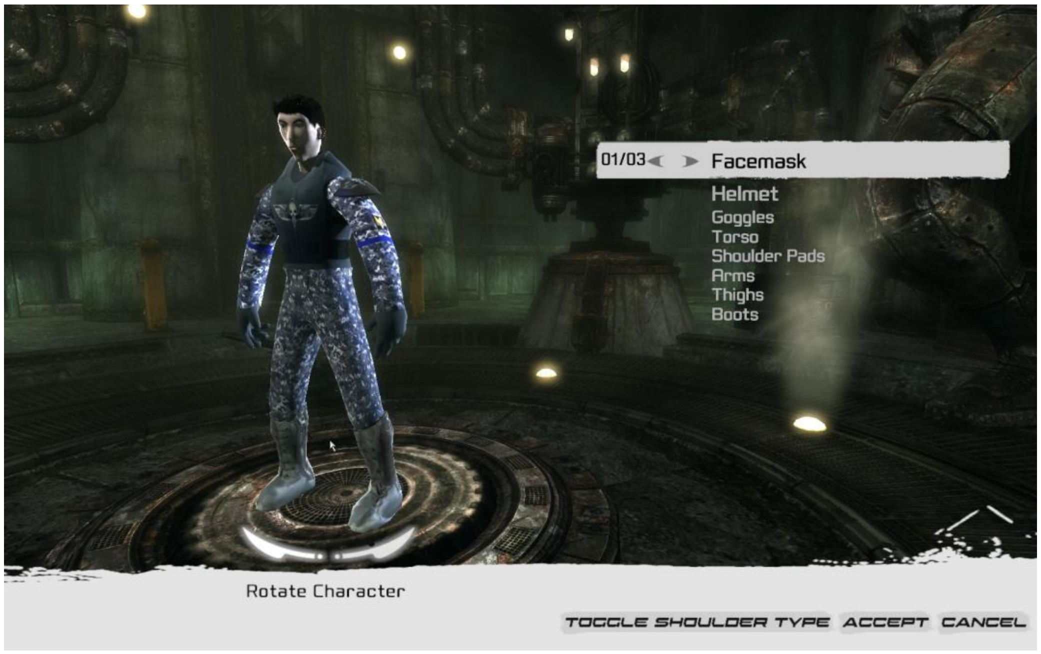
Evidence of the final asset working in game



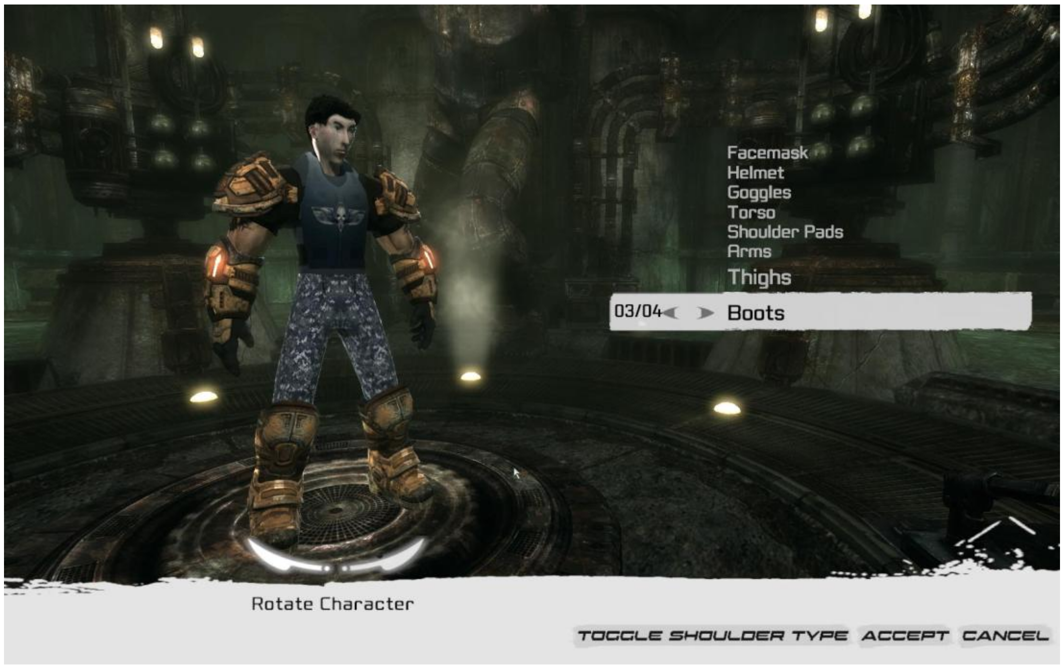
Commandos Team Leader, Crazy as hell. As in Wont Sleep or eat, Untill The Job is Done. Will happily jump off buildings.   Hows this for an innovations project??

Illustration shows portrait picture working in game.

Evidence of the final asset working in game



Screenshot from Unreal Tournament. Showing finished Character.



Screenshot showing character with Unreal tournament arms and boots. (Fully interchangeable.)

Summary
If you failed to achieve your goal(s), why weren't
they achieved?
The model and textures worked perfectly in the character select screen, and inside the editor. However in-game, the transparency maps didn't work correctly for the characters hair. I could not figure out why this happened. I believe more research into the opacity map settings would reveal it. I don't think this was too much of a problem, because I was being very ambitious in creating hair for my character anyway. Most in-game characters didn't have hair, only helmets or texture of stubble.

Additionally I would have like to create my own level if I had time to do so, for this project. I will continue to create one in my own time, for fun.

I've learned that I need to keep things simple, especially at the beginning stages. It is very easy to model into the 1000's of polygons, however this makes laying out uvs much more difficult and time consuming and therefore texturing and rigging become much more complex.

How would you do it differently if you did it again?
If I did it again, since I now know how to do everything. I would spend more time modeling the torso and body parts, increase their level of detail. I would also decrease the number of divisions in the head mesh and use vertex normal's to fake detail. Rather than having such a high polygon count. I would also like to spend a lot longer on creating more detailed normal maps using Zbrush and higher detailed diffuse textures for the body in Photoshop. Other wise I wouldn't really change my process of working. I planned effectively as I could, with limited information and succeeded in creating a finished asset.

Critical analysis of my work.

What I have learned in this project has changed how I create normal maps and has enabled me to create better textures for my major project.

In addition I have made great progress in understanding how rigging and local rotations of joint axes work. Also in completing this project with a low polygon count in mind, I have learned a lot about using face and vertex normal's to create a believable model but using a much simpler mesh. This has really helped me keep my render times to a minimum for my major project unit.

This unit has also taught me a lot about game engines and the numerous restrictions in place to be able to use them. I believe by trying to explain my process has actually helped me learn that much better. For example I had a few problems with team colours and getting the right emissive colours to work properly, but I figured it out as I was writing the report. So I decided to explain that part in as well.

I think that my specular, normal and diffuse maps could have been a lot better. I think in game they don't look quite right especially because I mirrored the normal map on some parts which gives a noticeable seam. Which I was rather disappointed with. Given more time to concentrate on these areas I think a more believable aesthetic can be achieved.

However I am quite pleased with the end result and the report because on numerous occasions I was very unsure of success. I may have had to rush some areas to get the project completed, but I managed to achieve my aims.

Given more time I'm confident I could create a character to an even more professional standard. Unreal Tournament 2007 took 3 years to develop with an entire team working on it. I'm Happy with what I managed to achieve in the relatively short space of time.

Appendix

This is a copy of an email sent to Mark Rein Head of Epic games, asking for more information.

Date: Tue, 4 Dec 2007 00:09:31 +0000 (GMT)
From: "Andrew Cosgrove" <and.cos@btopenworld.com> Add to Address Book
Subject: UDN developer network
To: mrein@epicgames.com

Hello Sir.

My Name is Andy Cosgrove, I am a final year student at Bournemouth university In the UK. I'm currently On the BA Computer Visualisation and animation course.

For my Innovations project I intend to create additional models for the unreal 3 engine using Maya/Zbrush and Photoshop etc.

And have them working in unreal tournament 3.

To help with this goal I was wondering if it would be possible to gain access to the "licence" area of the UDN development network. For more information on the unreal editor, than is publicly available.

My tutor Stephen Bell Suggested I try contacting You/Epic games about this.

My tutor is also thinking of trying to setup a games course at our University.
If you would like to speak to him to discuss this or verify who I am. His email address is

SBell@bournemouth.ac.uk

Thank you for your time

Best regards

Andy Cosgrove

From: "Mark Rein" <Mark.Rein@epicgames.com> Add to Address Book
To: "'and.cos@btopenworld.com'" <and.cos@btopenworld.com>
Date: Mon, 3 Dec 2007 19:33:50 -0500
Subject: Re: UDN developer network

```
Andy,

Sorry but we can't provide you with access to that.


Mark Rein
Epic Games, Inc.
```

```
Visit us at www.epicgames.com
```

UTCustomChar.ini additions.
Copy of the code that is added to this script file to enable the character to work

```
Parts=(Part=PART_Torso,ObjectName="Commando.Parts.Commando_Torso01",PartID
="0A",FamilyID="IRNM")
Parts=(Part=PART_ShoPad,ObjectName="Commando.Parts.Commando_XShoPad01",P
artID="0A",FamilyID="IRNM")
Parts=(Part=PART_Arms,ObjectName="Commando.Parts.Commando_Arms",PartID="0
A",FamilyID="IRNM")
Parts=(Part=PART_Thighs,ObjectName="Commando.Parts.Commando_Legs",PartID="
0A",FamilyID="IRNM")
Parts=(Part=PART_Boots,ObjectName="Commando.Parts.Commando_Boots",PartID="
0A",FamilyID="IRNM")
Characters=(CharName="Coz",Description="Commandos Team Leader, Crazy as hell.
As in Wont Sleep or eat, Untill The Job is Done.  Will happily jump off buildings like a
depressed lemming.       Hows this for an innovations project??
",CharID="0A",bLocked="false",Faction="Ironguard",PreviewImageMarkup="<Images:
Commando.HeadShots.Commando_Coz>",CharData=(FamilyID="IRNM",HeadID="0A
",TorsoID="0A",ShoPadID="0A",bHasLeftShoPad=true,bHasRightShoPad=true,ArmsID
="0A",ThighsID="0A",BootsID="0A"))
```

Web References

[na]. 2008. ActorX plug-in for Maya 8.5.
[na]. Available from:
http://udn.epicgames.com/Two/ActorX.html. [Accessed 23rd
February 2008]

[na] 2008. Normal mapping definition
[na] Available from:
http://en.wikipedia.org/wiki/Normal_mapping. [Accessed 24th February 2008]

[na]. 2008. Unreal reference Skeletons. Available from:
http://udn.epicgames.com/Three/UT3Mods.html#Characters/ [Accessed
23rd February 2008

[na] 2007 Unreal wiki reference available from :
http://wiki.beyondunreal.com/wiki/Solid_Snake/ImportingSkeletalMeshesIntoRob
oBlitz [Accessed 23rd February 2008]

[na] 2007 Forum discussion on creating custom characters. Available from
http://utforums.epicgames.com/showthread.php?t=585728 [Accessed 23rd
February 2008]