Models for Games

Notes on model creation workflow
using A:M and 3D Gamestudio

About this document

The main target for this 'manual' is myself.
This document is nothing more than a collection of tips, techniques and best practices I've put together along years of A:M modeling.

My experiences exporting models for games is far more recent, but I decided to also write down here all the tips and information I may find useful. Most of the examples involve modeling in A:M and importing this model into Gamestudio, but the techniques shown here may be useful for other applications too.

As I feel that some bits information are starting to fade from my cluttered memory, I decided that it was time to write it all down, and once done that, perhaps share this with other people.
This is not a tutorial for beginners. I tried to be as clear as possible on the explanations, without getting too technical, but I wanted to focus in information that you don't normally find somewhere else. I won't explain how to draw a spline or how to save a file here: I am just describing some specific methods and techniques that have proven useful to me and perhaps others.

Please forgive me about the rude layout and quick language. I'll try to improve this as I write on.
Should you have any suggestions or corrections, please e-mail me.
emilio@moscafilms.com.br

Links

Hash Animation Master (A:M)
www.hash.com

AMXTex .X exporter for A:M
www.obsidiangames.com

3D Painter – paint application and plug-in for A:M

www.3dpainter.com

3D Gamestudio
www.3dgamestudio.com

Version History

(2006-08-03) Started general layout and overview
(2006-08-04) Added UV Mapping
(2006-08-05) Added Double Sided Geometry

MODEL WORKFLOW

Overview:

Characters are modeled, rigged and animated in A:M.
Complex models with eyeballs, teeth, fingers and facial expressions should be around 1000 patches.
Details can be removed for lesser characters, like eyeballs, fingers, toes and some splines, for ideal complexity of 400 to 800 patches.

Models are exported to DirectX format using the AMXTex plugin from www.obsidiangames.com, which also exports the actions as .X animations. Models are then imported into Gamestudio using MED (3dgs own Model Editor).

AMXTex can export models at different polygon resolution. At 1 polygon per patch, a 1000 patch model actually generates about 2000 triangles. At 4 polygons per patch, the same model produces about 8000 triangles. This may be useful for creating different levels of detail, for options inside the game, or for LOD.

AMXTex can export A:M CPWeights. Even when not using weights directly in the model, the 'intermediary' vertices generated by AMXtex at 4 or more polygons per patch are automatically weighted.

Gamestudio doesn't support vertex weights right now. Exporting 1 polygon per patch produces the best animation . While 4 polygons per patch produces a nice soft model, it usually shows undesired animation, mainly on joints with fan bones.

Thru experimentation, the best result when exporting 4 polygons per patch is to use the Export CP Weights option, and having Gamestudio's MED .X importer to handle (eliminate/ignore) the weights in the .X file. See example below.

Example 1:

Model exported at 1 poly/patch. CP weights exporting doesn't affect the joints.
(This model uses knee fan bones)

Figure 1

Example 2:

Exported at 4 polygons per patch without CP weights. One knee is deformed but not the other one. This is because the knee fan-bones are being assigned to new vertices in a different way.

Figure 2

Example 3:

The same model exported at 4 polygons per patch and **CP weights**. Gamestudio eliminates the weights. This result is acceptable for 4 poly/patch exporting.

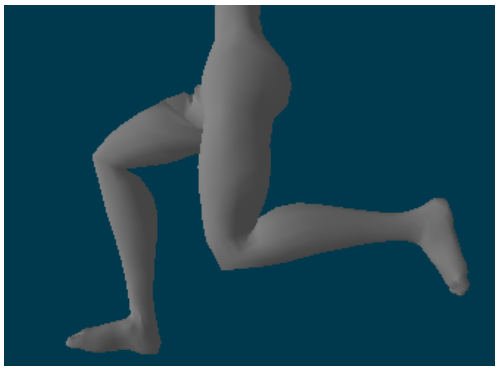


Figure 3

Modeling in A:M

This section is intended as a guide for creation and optimization of A:M models for Gamestudio, yet it contains some tips that could be useful in other situations.

Model size and measurement units

While A:M uses metric system by default, Gamestudio uses relative units called Quants. There is not a 'right' relationship between quants and real measures, but most models included with the engine measure about 64 to 80 quants, so they were most probably using a correspondence of 1 quant = 1 inch.

It is possible to use a custom relationship between quants and centimeters. However, it's recommended to approach Gamestudio standards, because of the way the templates and other features are handled.

So, independent of which units we are using in A:M, real-size humanoids will usually import as 2.5 times the size of Gamestudio standard models. If we want to get close to this size, we can import them with a scale factor of 0.4 (See *'importing models in MED'*).

Another way, of course, is to scale models in A:M before exporting.

Tip: When modeling at life size, setting the A:M grid to draw lines every 16 inches (40.64 cm) will reflect Gamestudio WED default grid (Low = 16, High = 8) or MED grid of 4/4 or 2/8.

The model origin and the Ground Plane

In A:M, it's a common practice in to model characters and objects 'standing' on the origin line. That is, the model's feet are about Y=zero.
In Gamestudio, this center point {0,0,0} is the model origin, and it's expected to be at about the geometric center of the entity and is used by the engine for some tasks.

1. Lighting. Entities in 3dgs are lighted after the shadowmap on the floor they are standing on. If the origin of the model is close the feet, there is a risk this origin will penetrate the floor, losing the lighting information for the model.

2. Collision. Even not using the templates, the engine uses a collision bounding box or hull located around the center of the model. So it's better to stick to its standards.

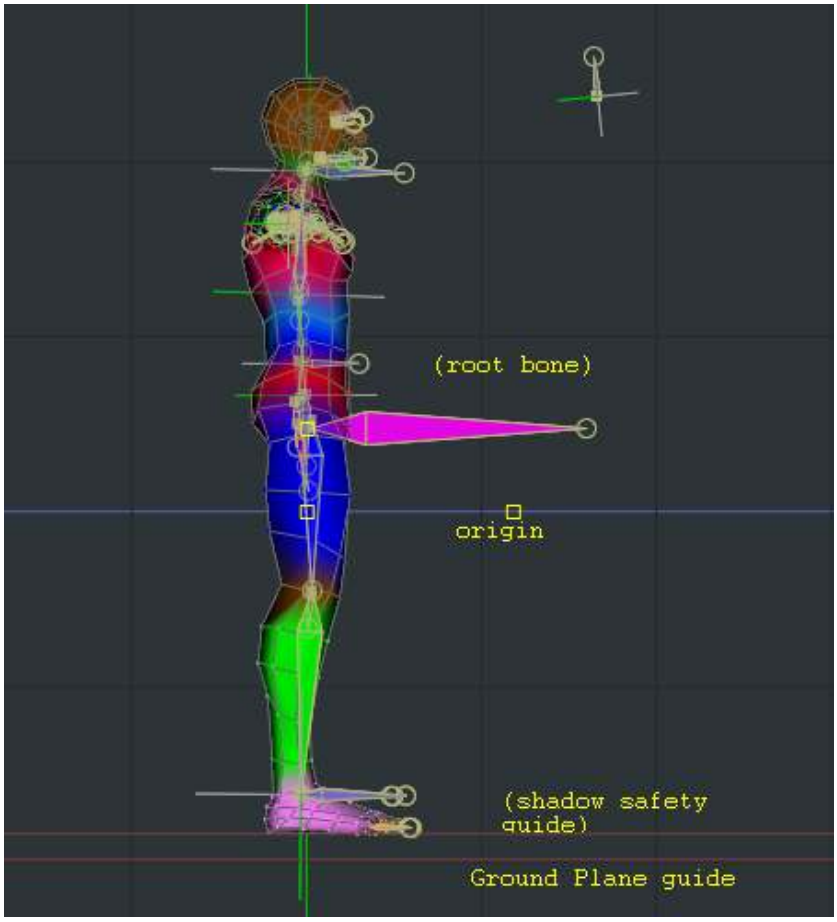


Figure 4: A:M model position and grid

As a good practice, I set up A:M grid spacing to 40.64cm (see Model Size above), then create a Ground guideline at the second grid line below the origin. Then I add another guide 6 cm (2 and half inches) above the ground plane and keep the character's feet or the object base slightly above this Ground guideline on most animations. This way, the grid matches WED grid, and the Entity is loaded on the right position from the start, and that shadows –which are rendered a short distance from the ground – do not intersect the model.

Figure 5: Nice shadows

Figure 6: Model too close to ground

Consult the Gamestudio manual about the right positions and proportions of a game model and animations (MED, Animation Rules, The Ground Plane).

5 point patches and hooks

5 pointers and hooks have always been a 'no-no' for A:M model exporting. This changed in A:M v13. Now, the core routines for polygon exporting make both 'features' not only acceptable but very efficient. A 5 point patch is exported at only a triangle and a 'quad'. Triangles are created to export hooks accurately too.

As for this moment, v13e has a small bug regarding texture exporting in 5 point patches and hooks, but I do believe this will be solved quickly. This bug doesn't affect 5-pointers when exporting above 1 polygon per patch.

Thus, given that the texture bug is fixed, 5 point patches are now recommended. However, I don't recommend the use of hooks: it's best to join the spline endpoint to another CP, deliberately creating a triangle that the user can keep under control.

About Textures

Textures quickly eat graphics adapter memory. Although newer cards may have lots of RAM, texturing for games should always take into account the optimization of texture space, with less unused image areas.

Also, in Gamestudio, like in many game engines, 3D models need to have all their texture information in a single image, and its dimensions should always be set to powers of 2, like 64x64 or 128x512. Old cards work best with sizes of 256x256. I like to work on a texture of 1024x1024, then downsize it for model exporting. This cannot be easily changed once imported in MED.

Color depth

The usual color depth in non-realtime 3D world is 24-bit. This means the image has 3 separate 8-bit channels for Red, Green and Blue (RGB) and is able to render millions of colors. 32-bit textures have the same color information as 24-bits, plus an alpha channel (see Alpha Channels below).

In the world of games, 16-bit textures are the most popular. They render only thousands of colors, but are good enough for most applications and use far less memory than 24 or 32-bit textures.

8-bit textures are indexed palette images, which consume little memory but have only 256 colors.

In Gamestudio, we focus in 16-bit and 24-bit textures. Textures should always be 32 or 24 bits in A:M, then they may be downsampled to 24 or 16 after importing the model in MED.

Alpha Channels, Overlay and Transparency

32-bit images are 24-bit images with an extra 8-bit alpha channel that can be used to define soft transparent areas of the model (or sprite). However, this is not as simple as it sounds. In Gamestudio, the mere presence of this alpha channel makes an object be considered Transparent.

> From 3D Gamestudio F.A.Q:
>
> Transparent entities - meaning entities with TRANSPARENT flag or with an alpha channel texture - can not write into the z buffer. Thus the engine sorts them according to the distances of their origins. Neverteless, transparency causes visible errors when objects intersect or when a concave mesh consists of transparent polygons. You just have to think a little to avoid those mistakes. For instance, make your 'X' from 4 connected sprites rather than 2 intersectings sprites, and separate a transparent mesh into convex parts.

Figure 7: Alpha Channels cause z-sorting rendering problems

If soft/partial transparency IS required in parts of the model, a *material* should be programmed and attached to this object, allowing the entity to write to the z buffer.

```
z-writing enable material

material zsort{
  effect="
  texture entSkin1;
  technique zs{
    pass p0{
     texture[0]=<entSkin1>;
     zenable=true;
     zwriteenable=true;
     alphablendenable=false;
   }
  }";
}
```

```
material right_alpha_mat
{
  effect =
  "
    technique right_alpha1
    {
      pass p0 {zWriteEnable=true; alphaTestEnable=true;}
    }
  ";
}

action right_alpha_acc
{
  my.material = right_alpha_mat;
  my.dynamic = off;
}
```

Enabling Z-writing is supposed to be slower than normal rendering.

Cookie Cut and Overlay

The fast and more efficient alternative to 'smooth' alpha transparency is simple Color Keying and Cookie Cut, which uses black or another color to create invisible parts of the decal. It is equivalent to the Overlay flag in Gamestudio. This flag causes all black parts of the texture to be rendered invisible. Alternatively, the RGB color at the upper left pixel can be used for this color culling instead of black.

Typical applications for Cookie Cut or Overlay are leaves, vegetation, holes, gratings. The edges of transparent areas may be coarse and hard, but it is faster than enabling z-writing, and can be used with 16-bit textures.

The choose of using or not cookie cut instead of Color in A:M doesn't affect the exported model. Only the overlay flag in the engine will determine if the black areas of a texture will be visible or not. But you can use Cookie Cut in A:M to preview how these parts will be rendered in real time.

Cookie cut and transparency often leads to the need of double-sided patches. See next topic for more information.

Double-sided geometry

Sometimes you will need geometry that renders both sides. Grass, clothes, sheets of paper, a flag…

The fact is, there's no such thing as double-sided geometry. A:M patches AND polygons are always single-sided. The ability to see backfaces in 3d applications is only a modeling feature, and it should be turned off for accurate feedback (by default, shift-6 in A:M), because this geometry will not render in the Game Engine, and worst: polygons that aren't part of a closed mesh will be considered Open Geometry and cause rendering problems with stencil shadows.

A double-sided 3D triangle is actually built from two different polygons whose normals point to opposite direction and that share the same 3 vertices. They are not considered an Open Mesh anymore: they are a closed mesh of two polygons.

Fortunately, this can easily done in A:M. The best practice for this is to create your geometry normally (leaves, foliage, grates, hair or whatever double sided parts the model needs), then rig and decal as usual. Create a group named 'DoubleSided' or something like that and add to this group all patches that should be exported as double sided.

When the model is ready to export, you should select the DoubleSided group and duplicate it in-place by copying and pasting all the group's patches. (Remember to set Paste/Extrude offset to Zero in preferences so you can paste the geometry in the same exact position). If Show Backfaces is off (shift-6) you should be able to 'see pathes from both sides' now.

Before exporting, you may have to rig the new geometry again, assigning the pasted CPs to the same bones as their original counterparts. This is an easy task, but can only be done manually.

Your model is ready for export. Don't worry about the duplicated CPs: they will be weld together later, after importing the model in MED (Read ***Cleaning Up in Med***).

UV Mapping a character in A:M

There are lots of personal preferences on this topic, but the main goal should be always the same: take maximum usage of the texture area, reduce the number of seams, and use more texture space for model parts that require more detail.

What I describe here is the method I use personally, which in my opinion puts together some good texturing practices.

Organizing model groups

I like to decal my model in 2 main 'stamps': Front and Back. Then, I add more stamps as I need them. To avoid unnecessary files, I create a single stamping 'Action' for each model and apply the stamps at different frames of the Action, which is based in CP (muscle) animation.

As this can create several groups and it may be hard to track in which frame I apply each stamp (so I can reapply again), I organize my model using a simple name convention: All the groups I use for texturing start with "UV", followed by the frame number, and then the View I am using to Apply the stamp. For instance, I create a group called UV05_FRONT which I will stamp in the Front view, frame 5, and another group called UV10_BACK, which will be stamped in Back view, frame 10. It's useful to have extra frames between stamps, because it makes easy to go back to the non deformed model and select geometry.

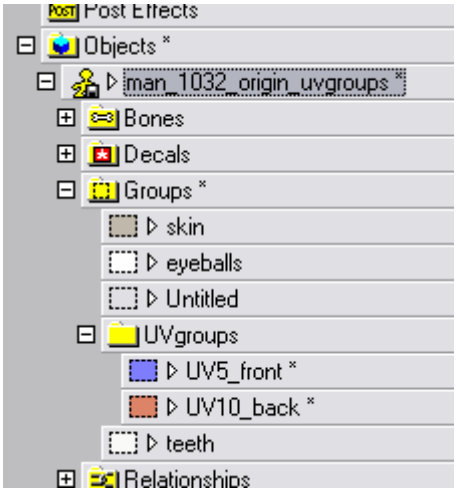


Figure 8: Organizing groups

After creating the groups, I define different surface colors for them, and start adding / removing patches to the groups until I'm satisfied. I use the Patch Select tool a lot. The image below depicts the groups UV05_FRONT(blue) and UV10_BACK (orange)

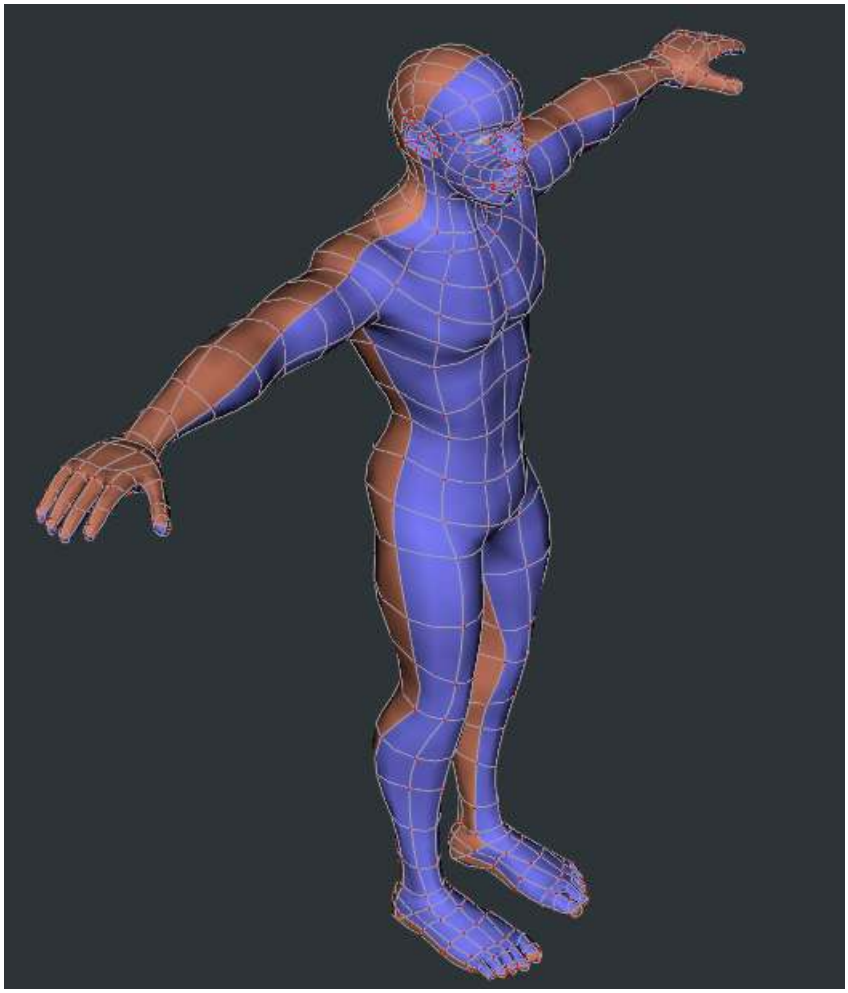


Figure 9: Front and Back groups

Note that I have created my model mesh in a way that I can easily split it in 2 stamps, including the hands and feet. Other modelers prefer to have the arms or head textured in different stamps.

Preparing the image

I like to use a tiled image template I made. The texture has also horizontal and vertical guides that mark the center.

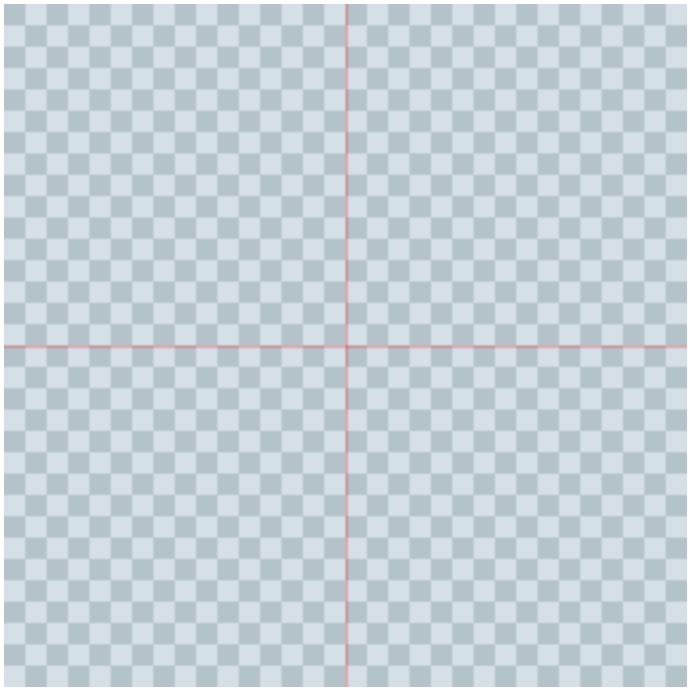


Figure 10: Tiled and quartered blank image

This, of course, may change. In this case I plan to decal all the front group on the left half of the image, and the back group on the right. But you could leave the upper section of the image for a weapon or prop. In any case, it's useful to create some pretty understandable guides in the image to make it easier to match the texture with the action.

The UV Action

First, I create some guides in the action that will mark the limits of the texture area I want to use. For the front and back stamps, I will need a rectangular area that is proportional to the left or right half of the texture. I use the grid to align my guides, so they are as accurate as possible.

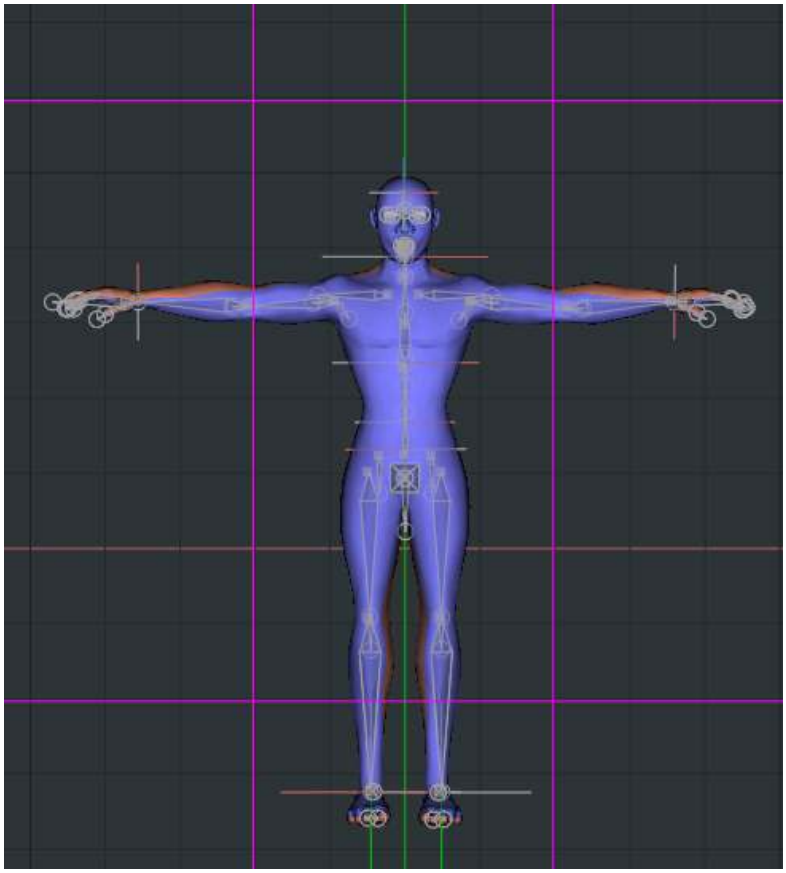


Figure 11: Guides define a rectangular area

Now there comes the manual work of flattening the model.

First, I go to frame 5 and try to distort the model in an overall way. I don't hide any geometry right away: I just select parts of the model and rotate them.

I avoid using bones. Actually, I turn off any rigs before starting to deform the model. It's all 'muscle' animation.
Since some parts need more texture information, these parts should be bigger in the decal. So I end up with a very big head and hands, and a smaller torso.

I often work on one side of the model and then select ALL CPs, Copy keyframe, select some CPs from the opposite side and PASTE MIRRORED. It works great.

After distorting the model, just scale Z = 0 and manually flatten / tweak overlapping patches until it's clean. When the FRONT frame is done, copy all muscle keyframes to frame 10 to prepare the decaling for UV10_BACK. Since the distortion is about the same, more than half the work is already done. Just some cleaning up should be needed.

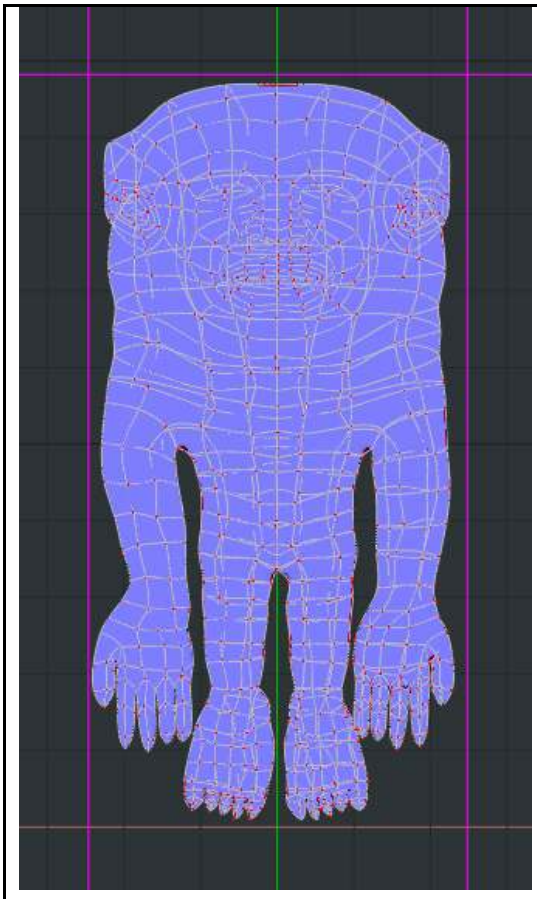


Figure 12: Front view at frame 5

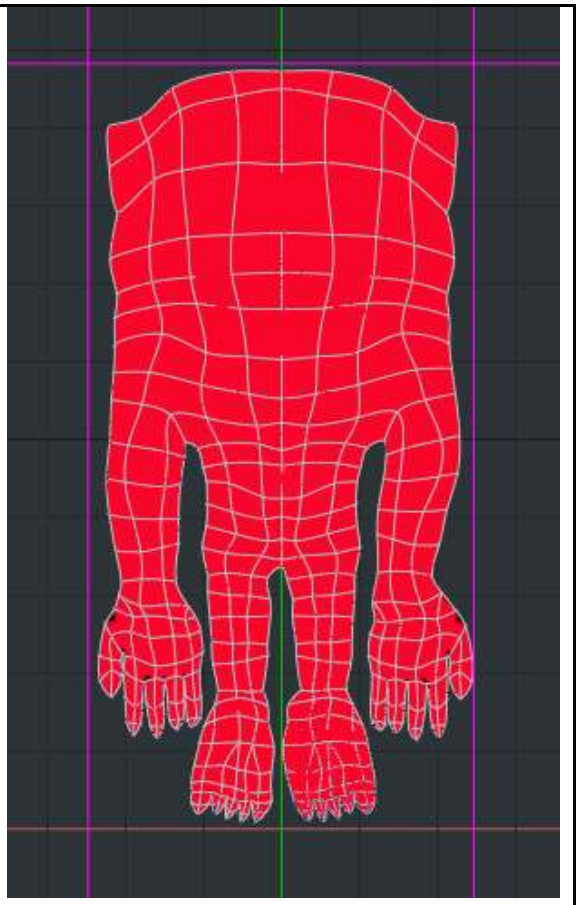


Figure 13: Back view at frame 10

Before stamping, there's an optional step that will generate better UV seams, with far less unwanted stretching at seams.
In both the Front and Back frames, the surrounding backside CPs should be pulled outwards, so the geometry edges isn't stressed. Hard to explain without a picture:

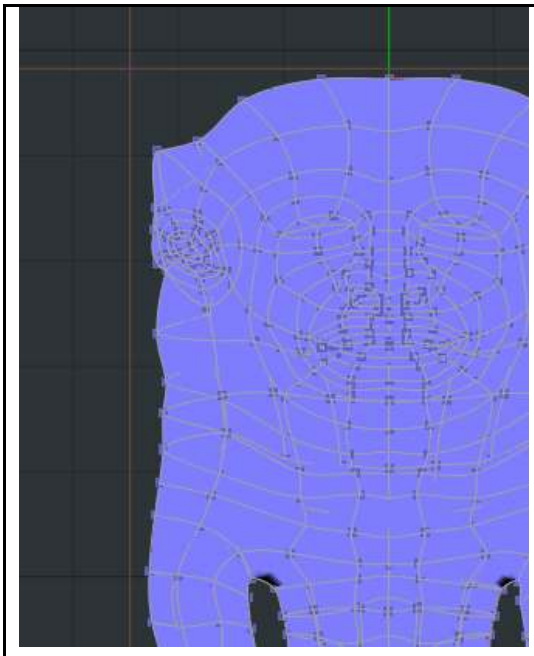


Figure 14: Before

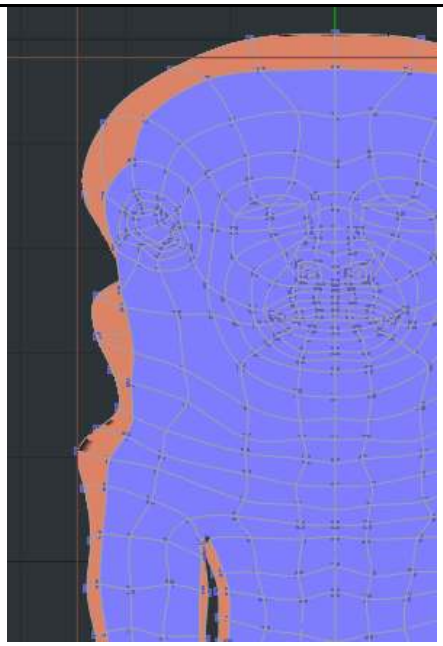


Figure 15: After

Tip: Before this last step, you may want to make 'backups' of your non-tweaked Front and Back frames. You can copy the keyframes in frame 4 and 9 respectively. If something goes wrong, you can go back.

Stamping

Once our action is ready, we only need to apply the stamps. Go to front view and frame 5, select the UV05_FRONT group and hide the remaining geometry. Then create a decal. Right clicking on the action window won't work: to create a decal in an action, you must right click the model in the Project Workspace instead.

Choose the image for the decal. It should show on the action screen. Then, use the guides in the texture to align the left half to the guides in the action.

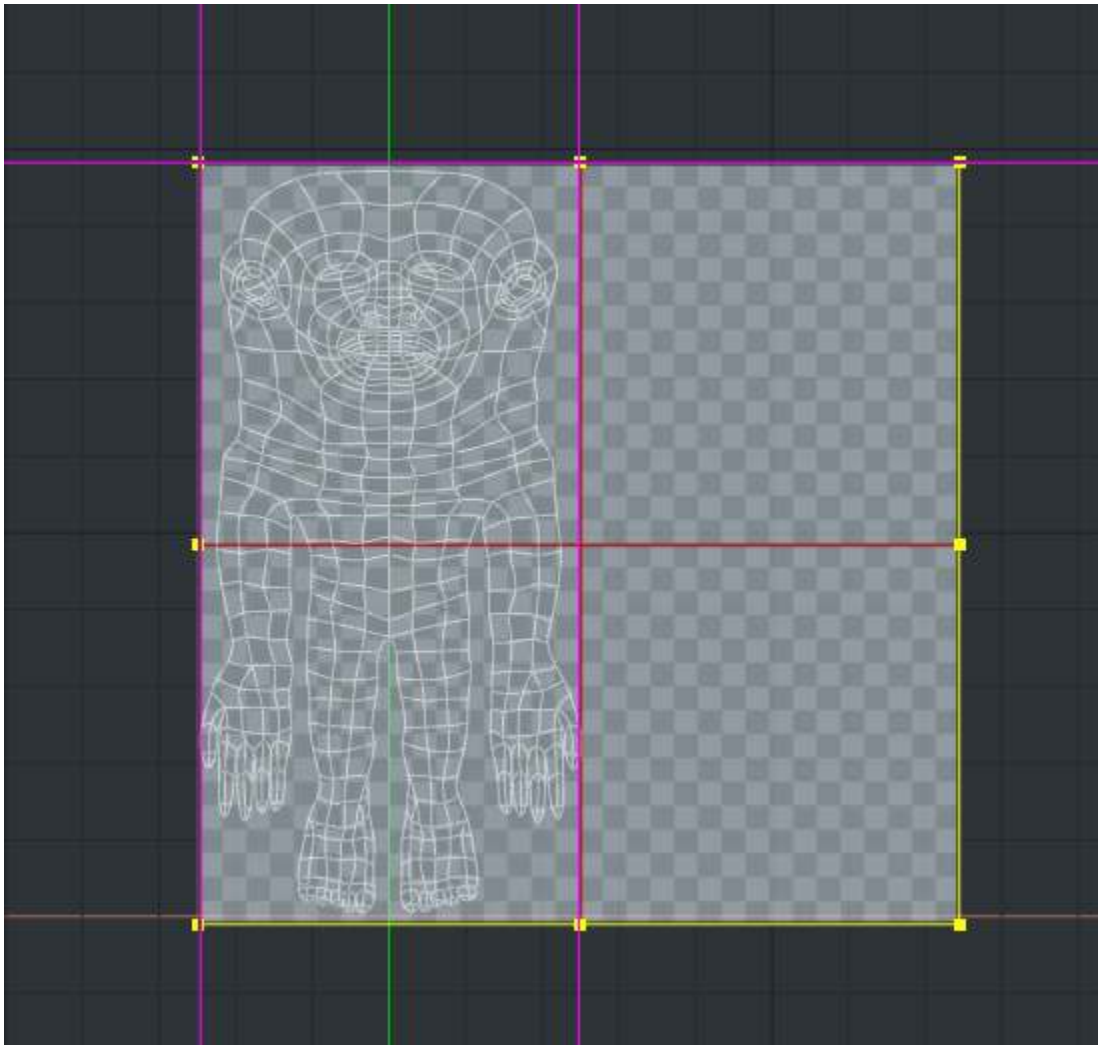


Figure 16: Applying the decal for UV05_FRONT

You may want to take your time in this important step. Zoom to the corners and markers of the image to align them to the guides. Also, it's best if you only scale the image proportionally, if possible.
When you are done, you may zoom the view to show all the model and APPLY the Decal in front view, frame 5.

Then for the Back stamp. Go to frame 10, in Back view, select your UV10_BACK group and hide the rest.
Double-click on the decal in the PWS to recall its position. You should reposition the decal it so its ***right*** half is aligned with the guides now. Now apply the stamp.

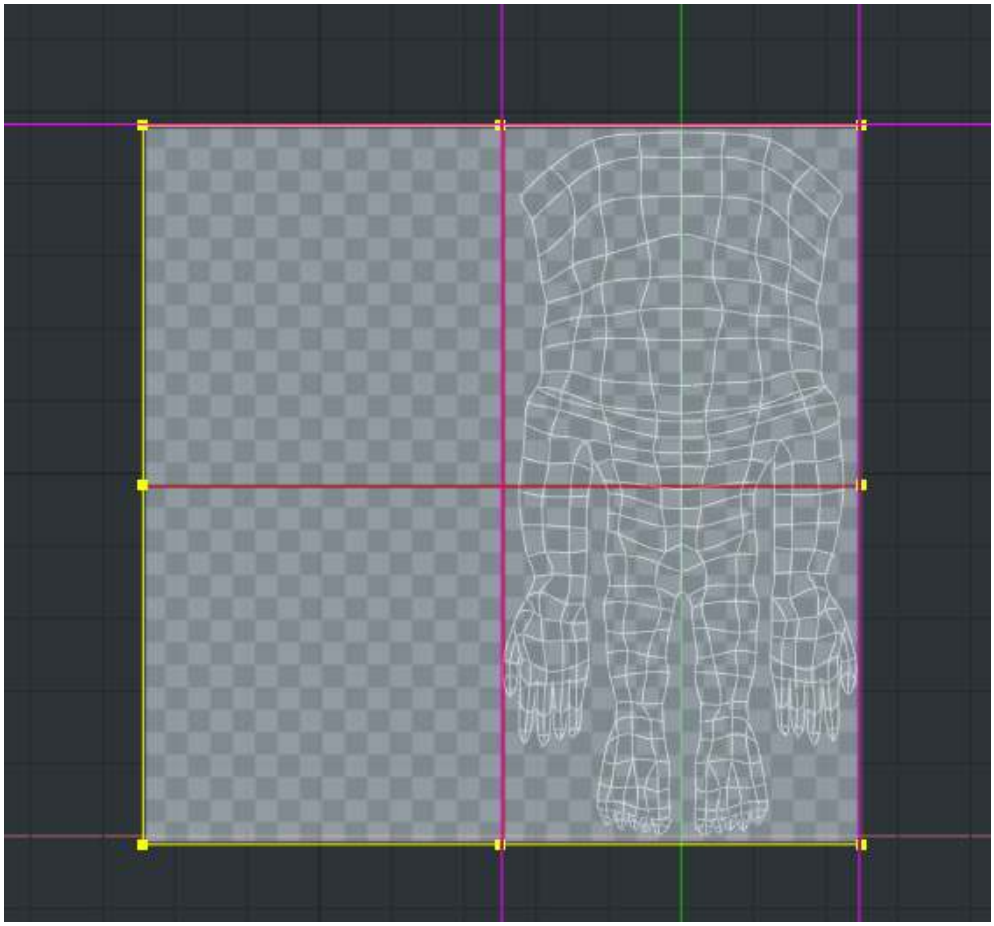


Figure 17: Applying the BACK stamp

Now it's time for checking your model. Check if all small patches at the nose, ears, fingers and other details have been correctly decaled – otherwise you may have to correct the CPs in the action and apply the stamps again.

When you are done, you should rename the Stamps in the PWS to match the groups.

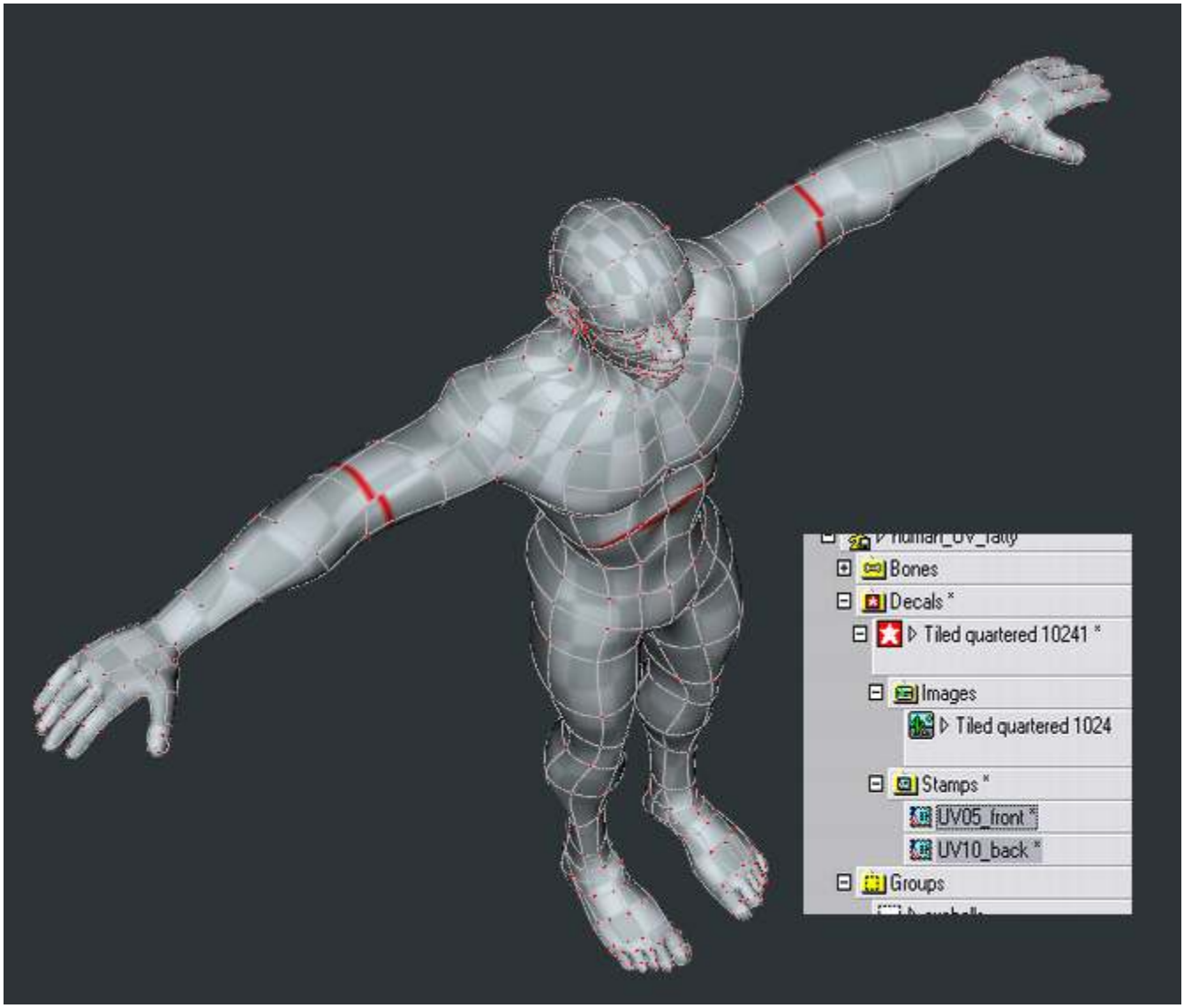


Figure 18: UV mapped model, with named stamps

You also may want to take a snapshot of both stamps, crop it carefully and keep it. You can resize it to your texture size (512x512) and paste it as a layer in Photoshop, to work as guide for your texture.

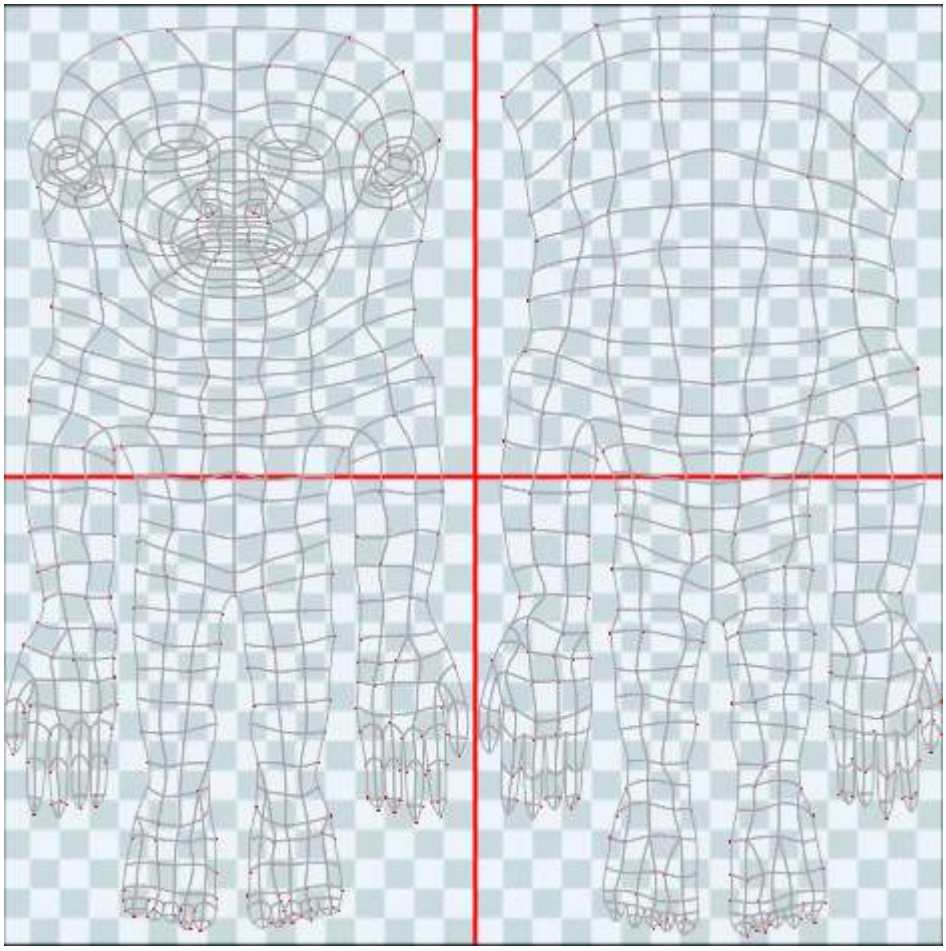


Figure 19: Decal snapshot

Eyeballs, teeth, hair, garments…

If there's more geometry to be UV mapped, you should create more *UVgroups*, and flatten the geometry in new frames of the *UVAction*. There's place in my decaled image for eyeballs or teeth. You can use naming conventions for organizing the groups, for instance: UV20_eyeballs_front or UV30_haircut_top, so you will remember in which frame and which view the stamp is being applied.

For more complex textures, you may plan ahead and 'tessellate' your texture by creating more specific guides in your blank image. Just try to use understandable proportions (for instance, only squares and 2x1 rectangles) so you can easily create action guides that match the sections.

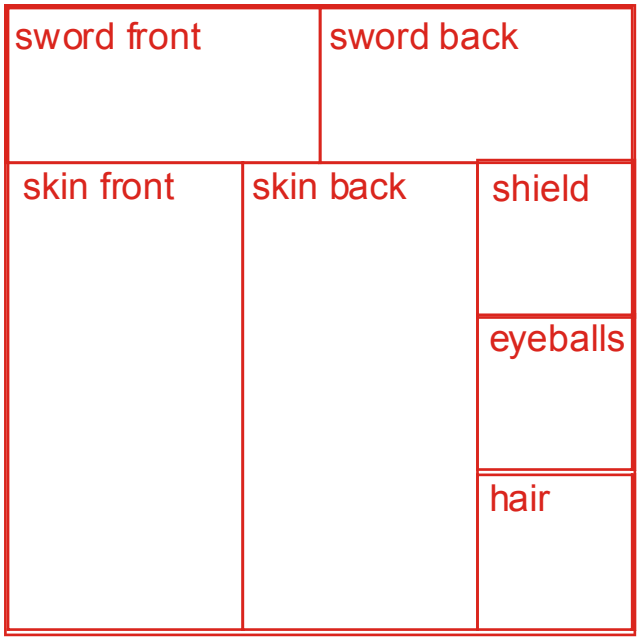


Figure 20: preparing a tessellated blank texture

You can also use the A:M limited UV editor to position and scale your stamps after decaling. But I don't recommend to individually tweak Control Points in the UV editor. It's best to leave the stamps the shape they are in the actions, so they can be restamped anytime. For instance, if you break or add geometry to a model, you will only have to reposition the new CPs in the action. The old stamp will be worthless then.

Advanced texturing techniques

Poor man 3d paint / texture bake

This is a powerful technique that can be combined with the texturing methods explained above.
With this technique we can stamp posed pictures all over the model, then 'bake' them in our optimized texture square.

I downloaded this hand from www.imageafter.com. I want to use it on my model, so I have to pose my model to match the image before stamping. The best way to do this is in an action. So, I create an action and add the image as a rotoscope on the TOP view. Alternatively, I could use a decal that displays as a rotoscope, but I prefer the rotoscope functionality, as the decal doesn't have transparency or locking.

I then position the rotoscope in the best starting place for the body part I want to decal.

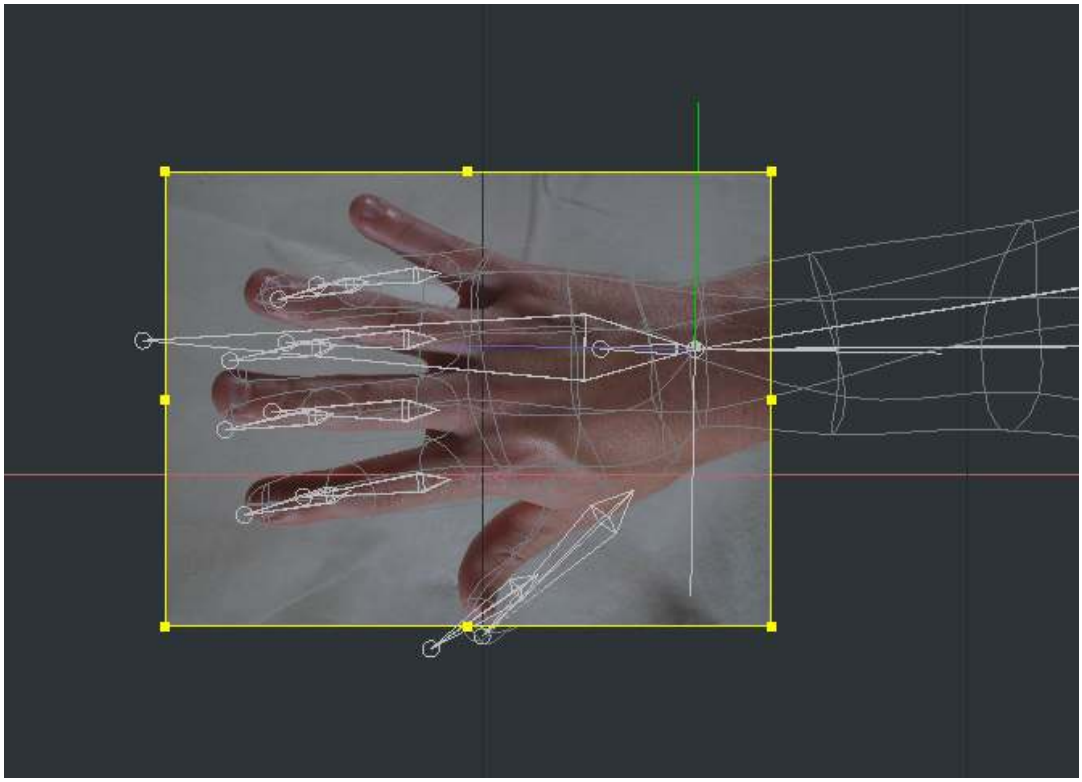


Figure 21: Hand rotoscope

I manually pose bones and then CPs so the model matches the picture. I prefer to do so in frame 10 of the action, so I can go back and forth to check on the geometry.

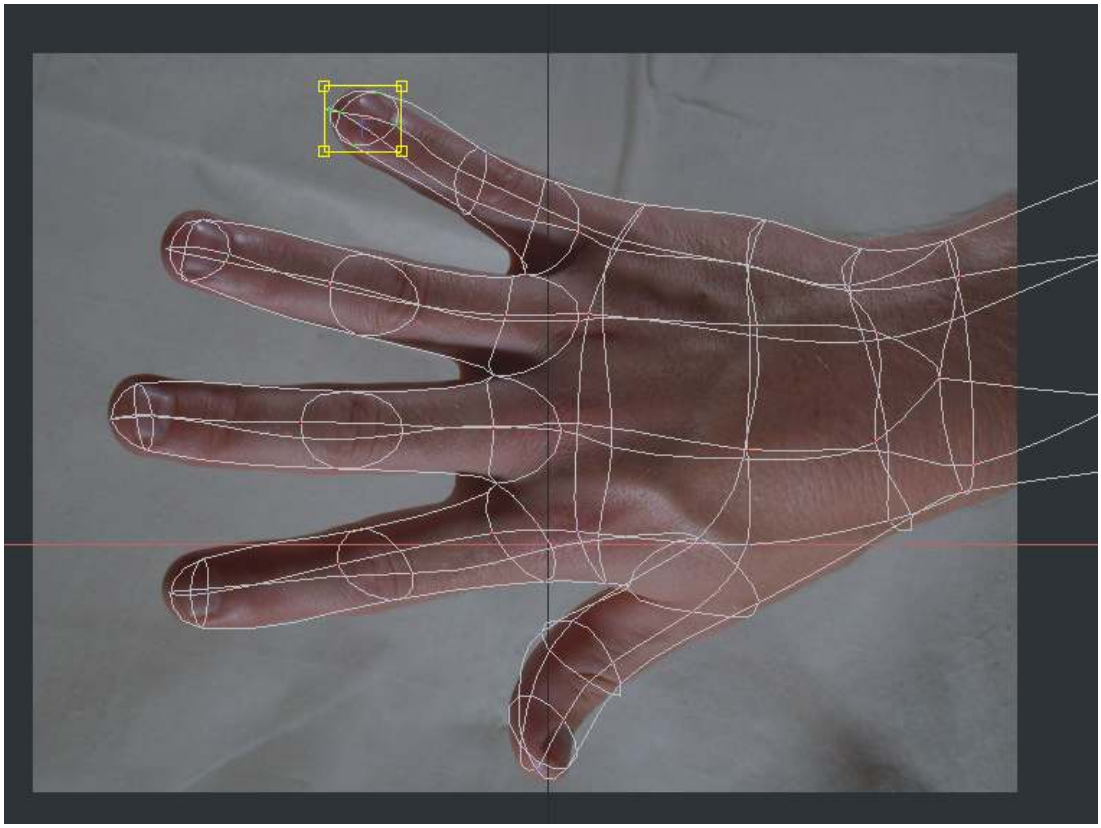


Figure 22: Posing the geometry

Then I add the same image as a Decal (by right clicking in the model in the PWS, not the action window). I match the decal position with the underlying rotoscope, then stamp the decal.

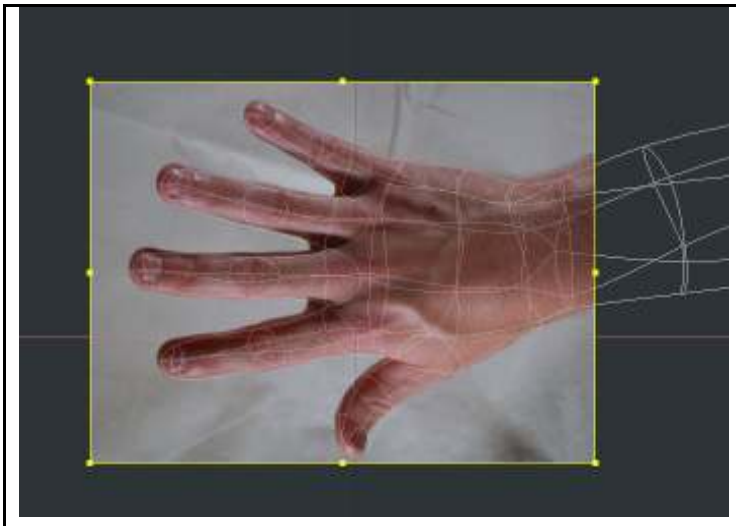


Figure 23: Positioning the decal

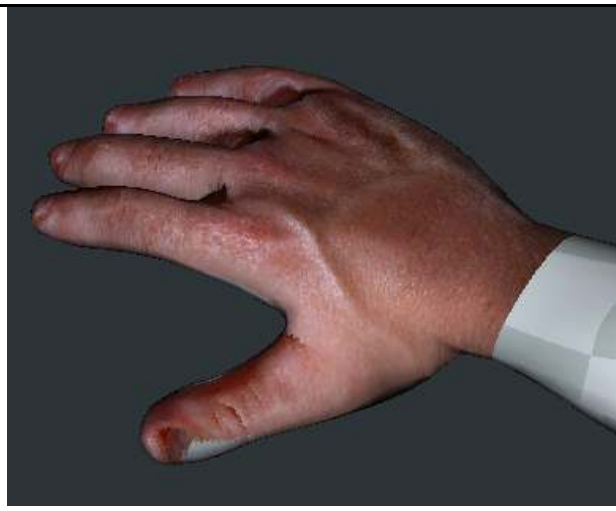


Figure 24: Decal applied

You can try this for the chest, arms, back, hand palms, feet. Even posed pictures that weren't intended as physical references will do well.

If the picture is a photo with some acute camera perspective, you may want to use a choreography instead of an action, and add the rotoscope to the camera, which is a bit tricky. First, you the rotoscope will try to take all the screen – you can rescale it with the mouse, holding shift, so it returns to its former ratio. Then, trying to position and scale the decal with the mouse will be impossible. You will have to edit the decal transform values in the PWS instead.
For this reason, it's far better if you keep the Rotoscope positioned at center. This way you will only have to worry about the decal scale when trying to match the rotoscope later.

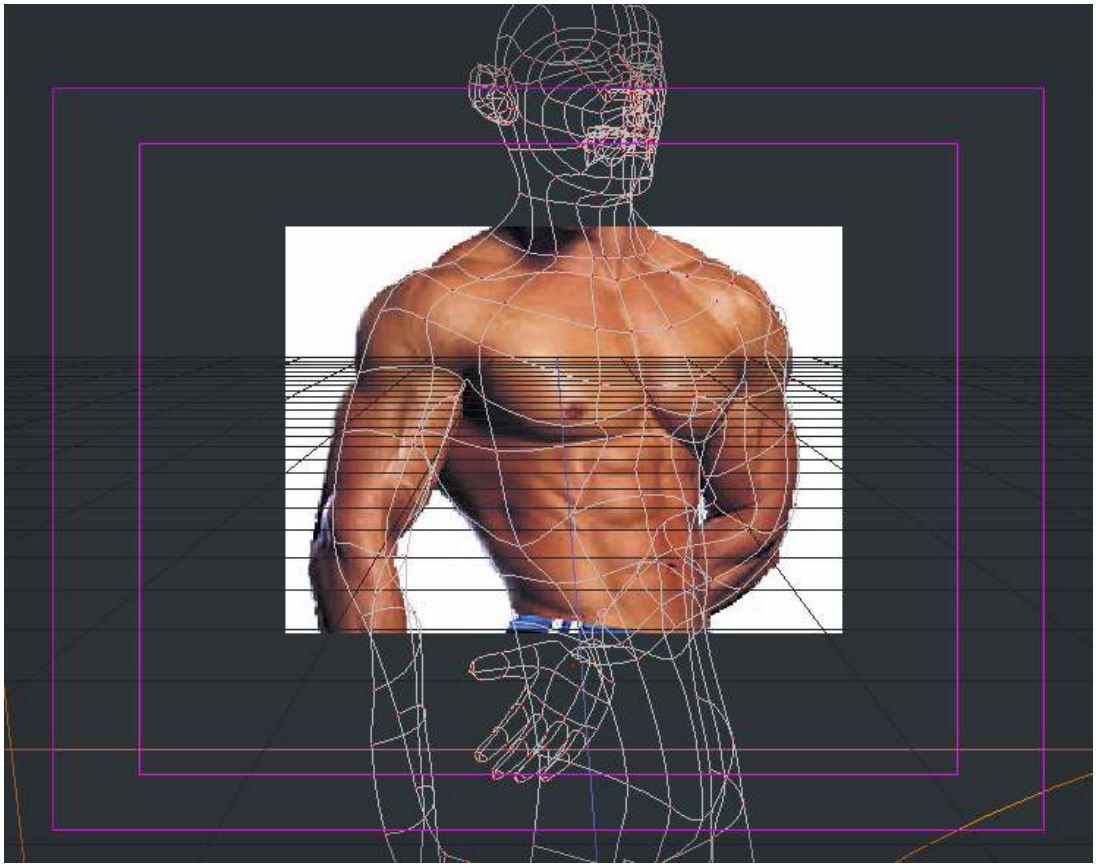


Figure 25: Matching a rotoscope in a choreography

We aren't looking for perfection in these stamps. We are just adding some reference textures and images. Given that the character is already rigged, it should be easy to get it to an approximate pose with bones, then complete the alignment with muscle animation. It is important, tho, that some splines match the anatomy. Particularly, we don't want the bellybutton or the back muscles too off center.

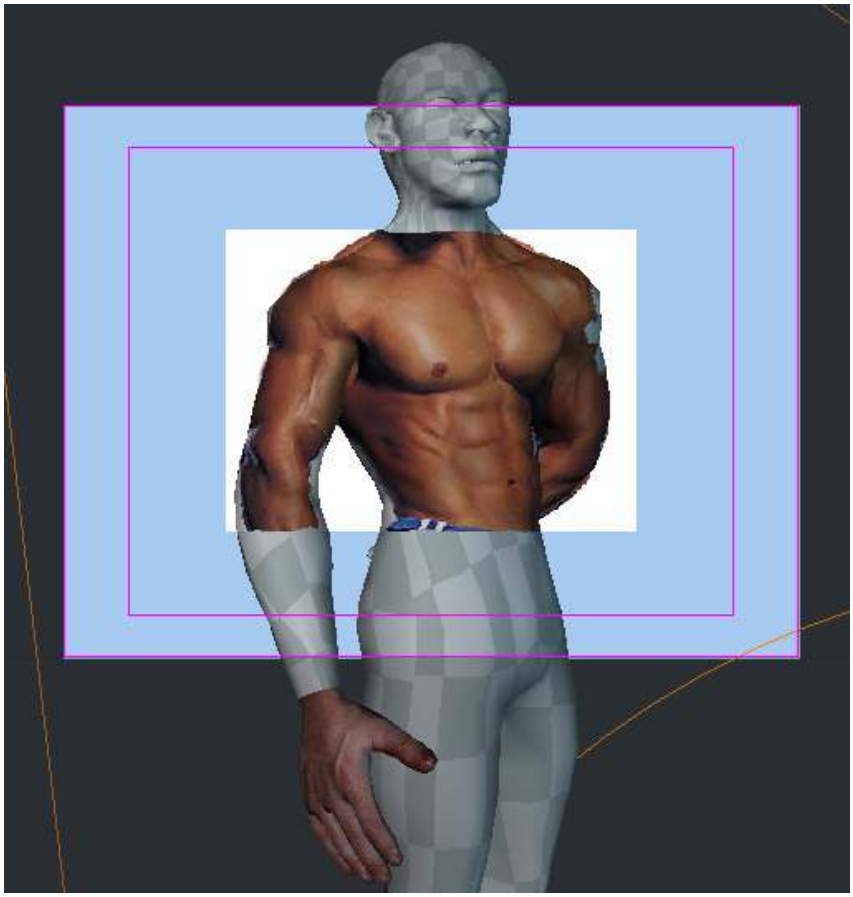


Figure 26: Chest and hand

Figure 27: Matching another pose

Figure 28: Shaded check

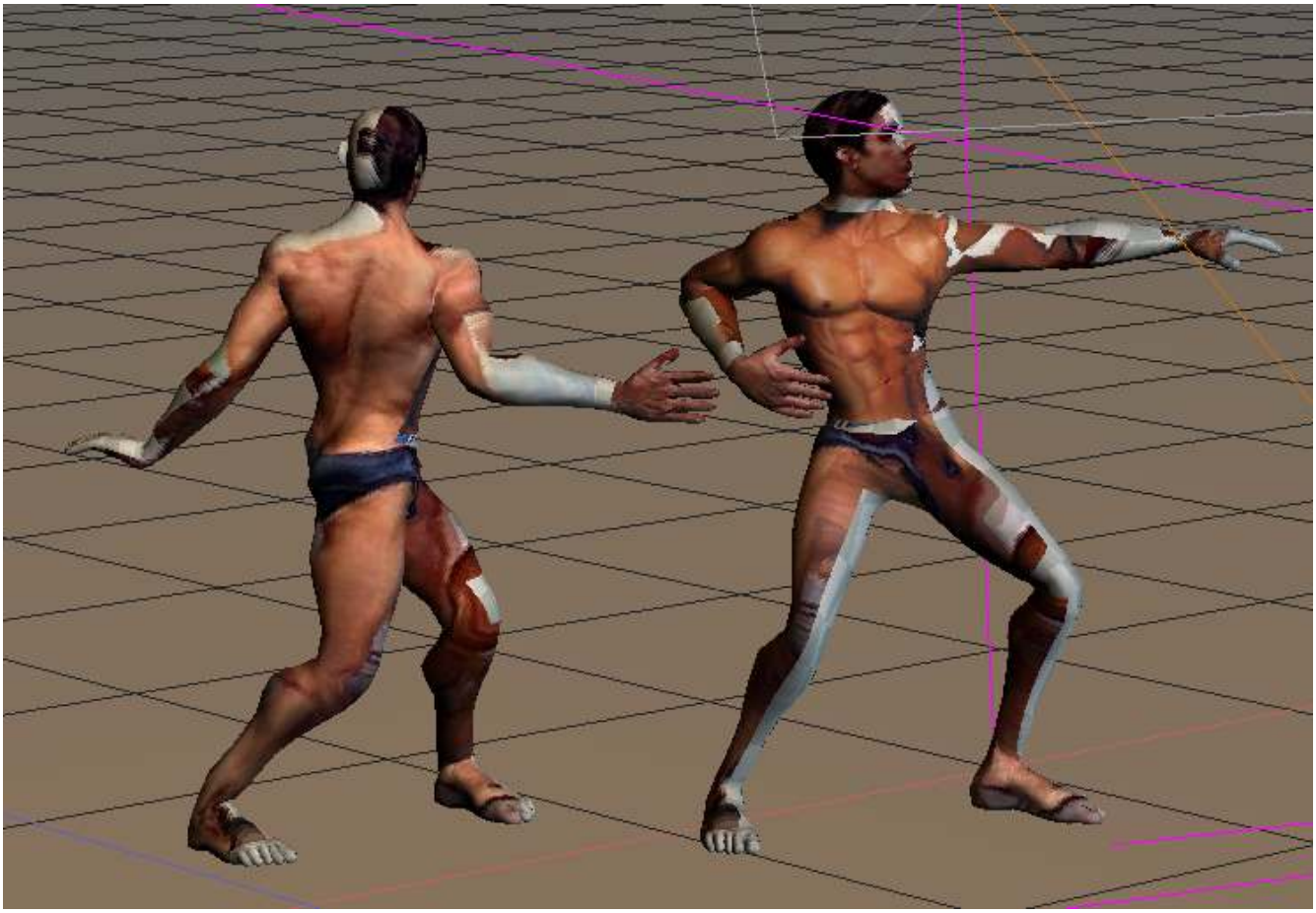


Figure 29: We created a monster

In every stamp, remember to hide geometry that is already decaled.

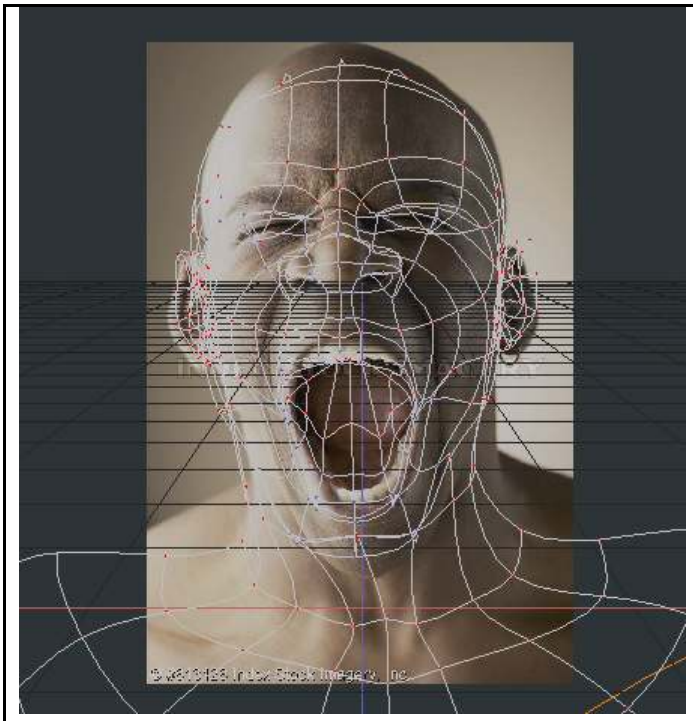


Figure 30: This was a weird choice for a face

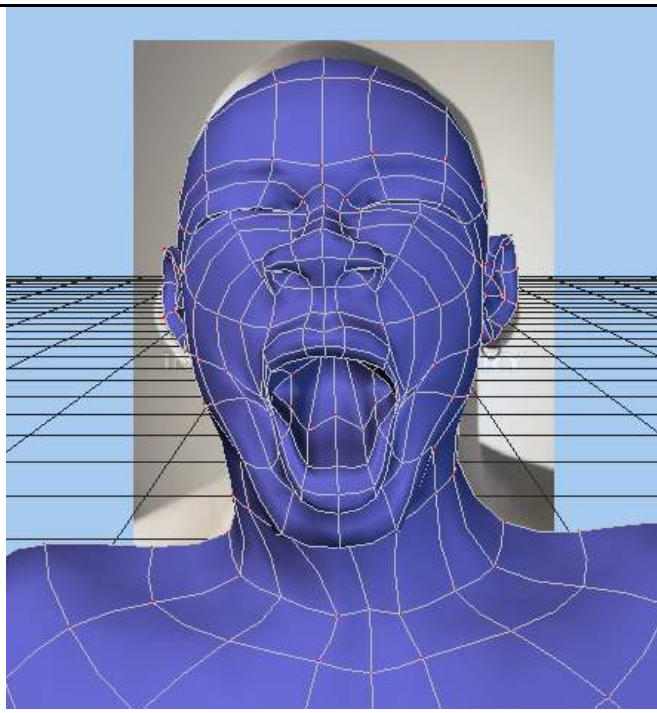


Figure 31: Matching geometry before applying

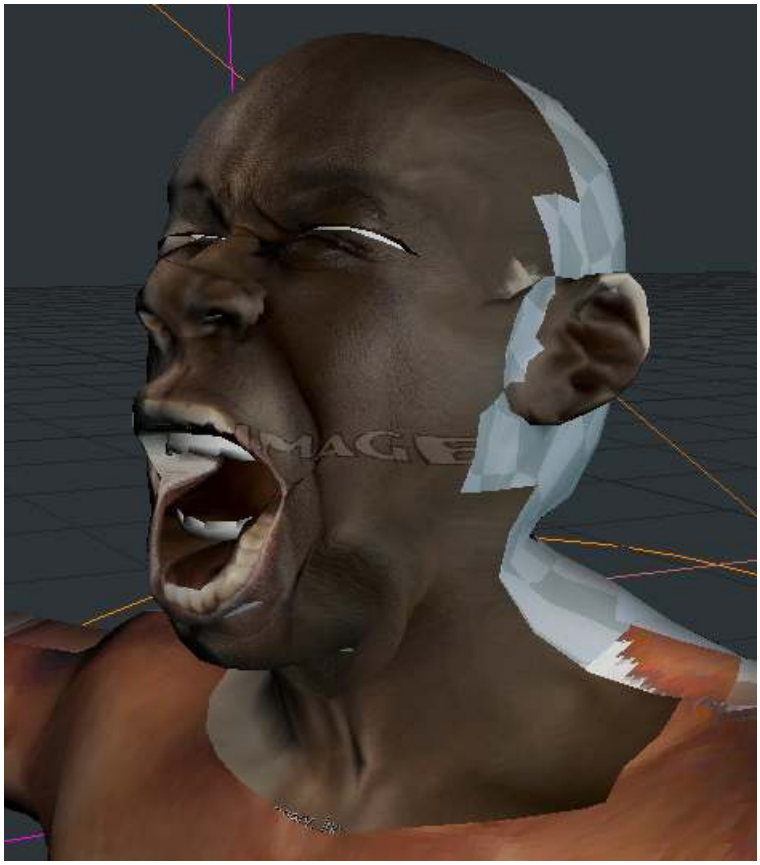


Figure 32: Yes, it IS a monster. Important parts match, however.

Putting it together

Now the funny part. What do we do with all those stamps?
We just have to open our UVAction and render it in the same frames we created the UVstamps. We can take a snapshot of a quick render, or better yet, we can render to a TGA file. Don't forget to hide geometry you don't want to render in each image.

Figure 33: UVAction - Front render at frame 5

Figure 34: Back render at frame 10

Since you kept a Decal snapshot (Figure 19), you may use it as a guide to paste these rendered images in place in Photoshop. Now you see why it's important that you don't mess with the Stamp in A:M's UV Editor.
Tip: you can set the 'wireframe' layer transparency so you can easily align the rendered flattened geometry to the 'wireframes'.

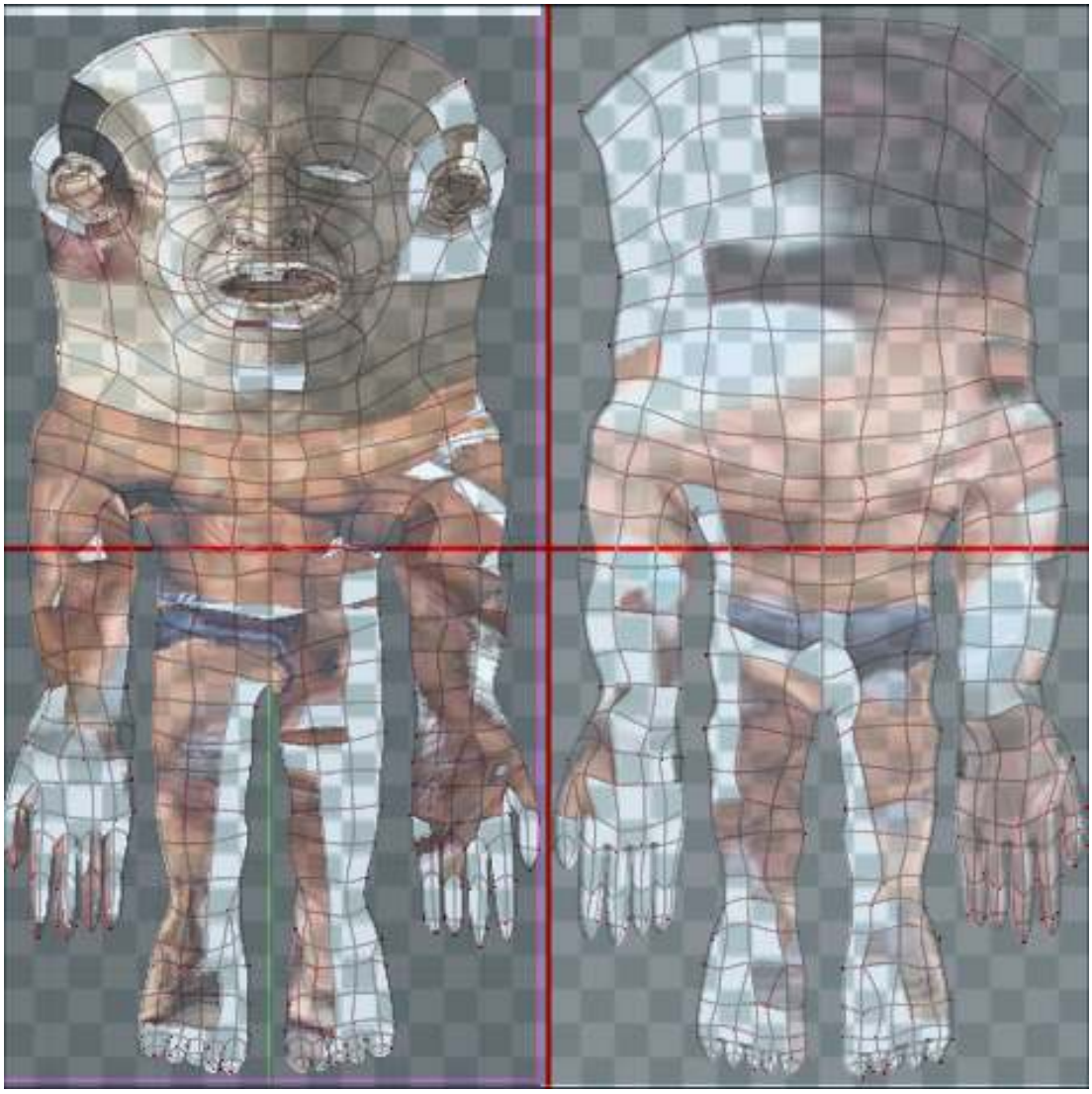


Figure 35: Matching front and back render with stamp wireframes

After pasting the render images in place, you can save the image and use it to replace the texture in A:M. Then, proceed to edit and retouch the image in Photoshop or any other image editing. For instance, you can copy the hand texture from one hand to the other, mirror and blend to the skin color.

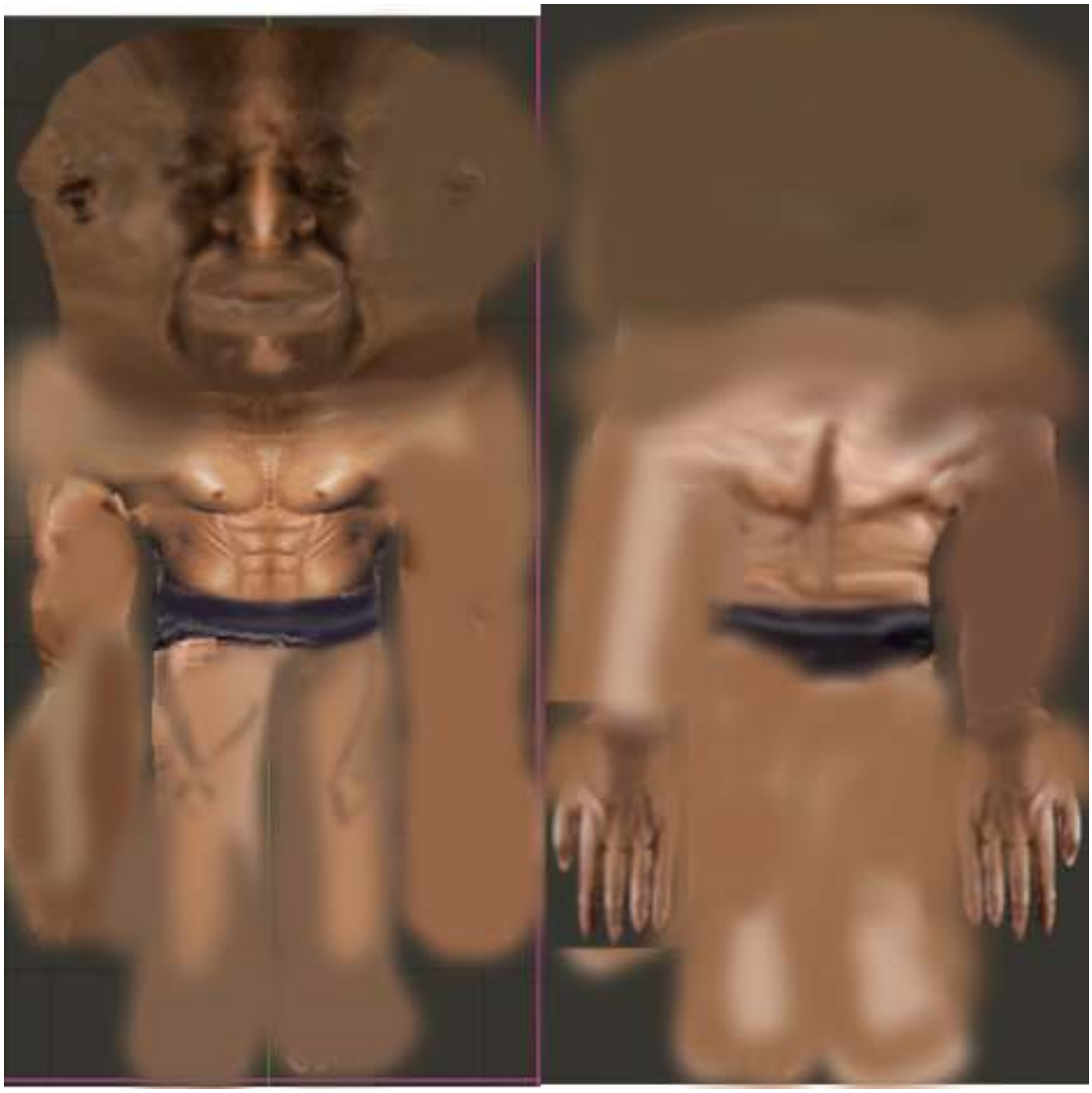


Figure 36: Blend in progress

I will not go into further details here. It all now depends on your ability with a brush to clone, smooth, mirror, equalize or otherwise blend images together. A last tip: images may be different in color, tone and lighting. If you are going to use this technique with a lot of images, it would be better to try and equalize them in your photo editing application *before* using them as decals.

Patch Images

Adding images to a group in A:M is a powerful texturing feature. However, the advantages and ability to use this feature for game models depends on the Game Engine.

3D Painter

{
Coming soon:
-Painting in 3d painter
-Using 3dpainter to blend textures
-Smoothing UV seams
}

Rigging and Animating

{
-animating for games
-simple rig
-joints, targets and constrains
-eyes, mouth, speak and expression
}

Exporting Models with AMXTex

This is a snapshot of AMXTex exporter window. These options are optimized for importing into Gamestudio, both for 1 or 4 Polygons Per Patch.

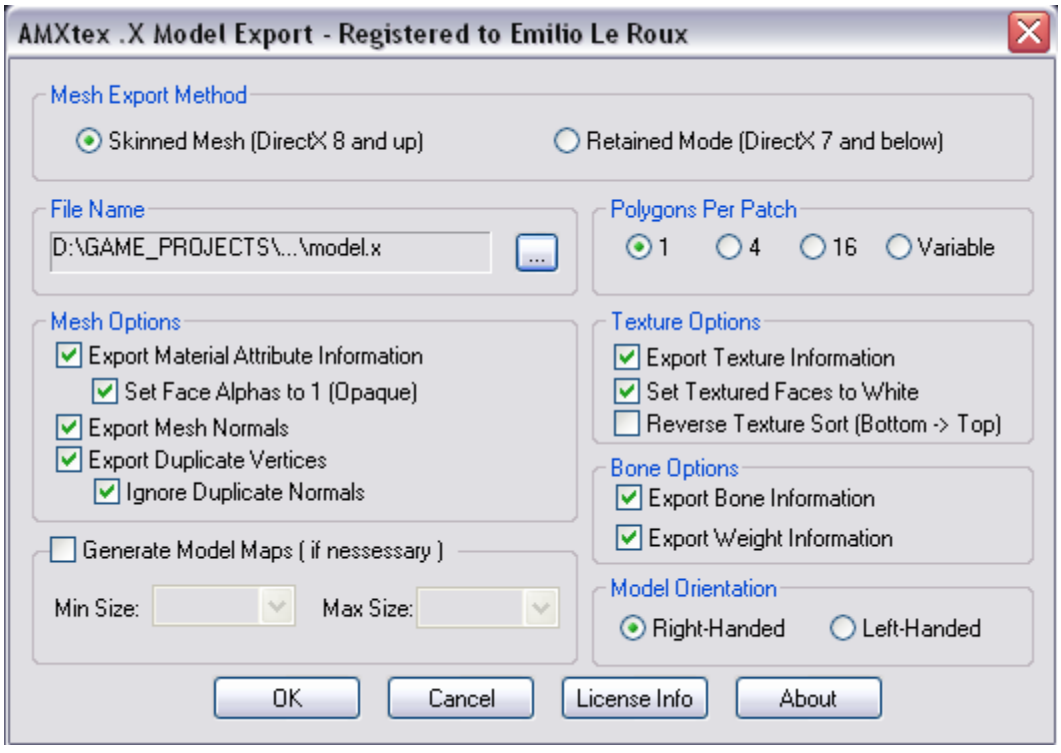


Figure 37: AMXtex Model Export dialog

Notes: currenty, Model Orientation doesn't seem to be working correctly in AMXTex. So far, Right Handed seems to be the right way to import into MED with the options below.

Exporting Animations

Animations are done in different actions in A:M. Only skeleton, and poses/relationships that animate model bones, should be animated.

Animation time in Gamestudio isn't tmecode based like in A:M. Thus, actions doesn't have to be created in real time, and they should be manually time-compressed in A:M before exporting to avoid creating excessive animation frames in Gamestudio. This is easily done by shift-selecting and collapsing the ***Bones*** and ***User Properties*** folders in the action model shortcut, selecting all keyframes and dragging the end handle to compress them in time.

Note that the project must be saved and the action closed/reopened to automatically update the Action **Duration** before exporting.

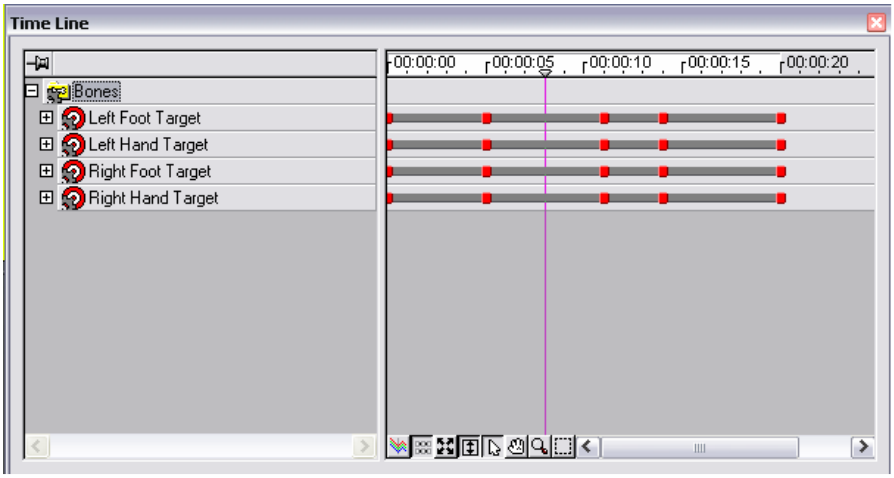


Figure 38: Full-length animation in A:M

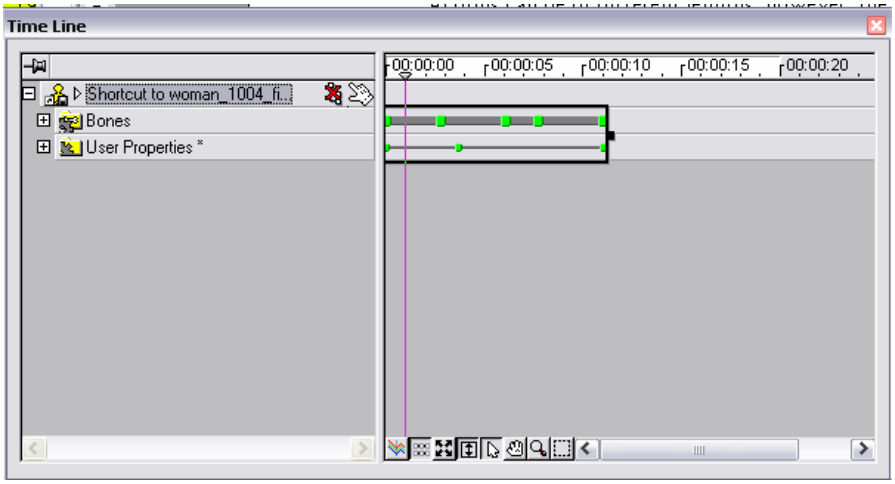


Figure 39: Compressing keyframes in A:M before exporting

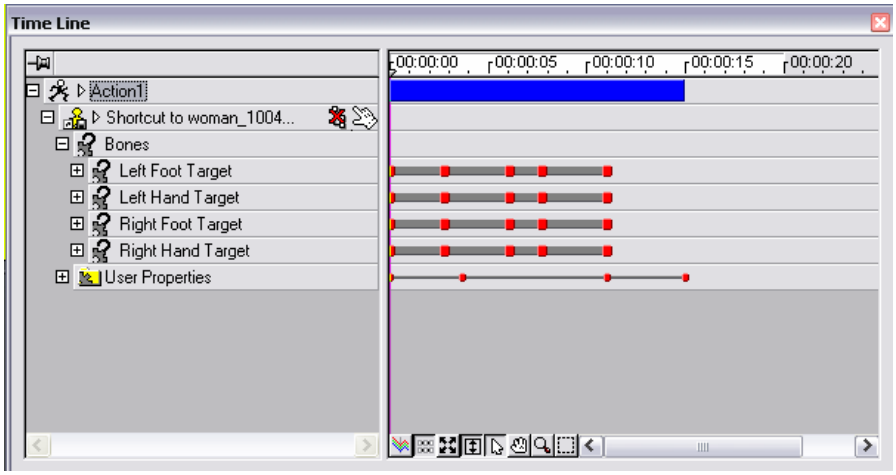


Figure 40: Action duration must be updated before exporting.

NOTE: there is a small bug in Gamestudio importer that replaces the last animation frame in each scene with a copy of the first animation frame of the scene. In repeating cycle animations, like a Walk Cycle, this last frame only needs to be deleted (in MED) but for non-cycle animations, an extra frame should be created in A:M so it can be eliminated later.

Exporting in AMXtex is straightforward.

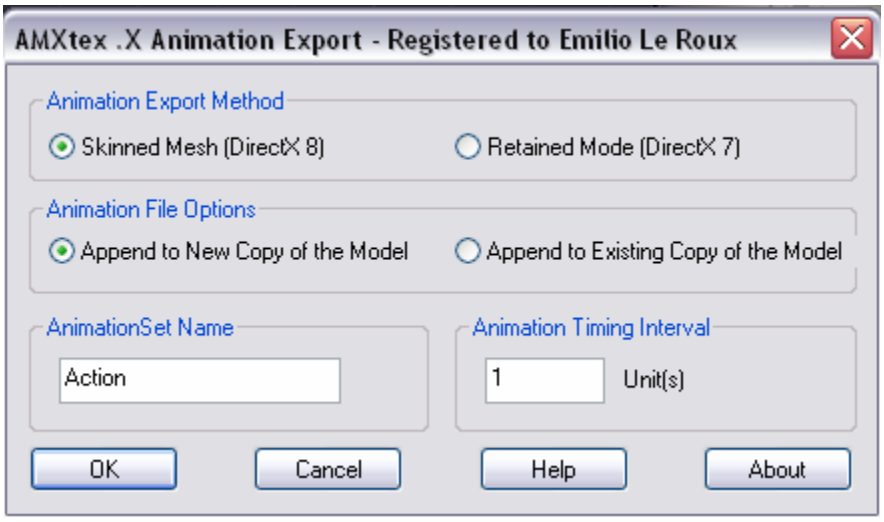


Figure 41: AMXtex animation exporter

Since all animation scenes should be contained in a single model, these steps should be used:

1. Export model with AMXTex model exporter
2. Export first animation with AMXTex, using 'Append to New Copy of the Model"
3. Export further animations with AMXTex using 'Append to Existing Copy…"

Importing models in MED

Gamestudio's Model Editor (MED) handles importing of AMXTex .X files. From the .X Import Dialog, select only the mesh, the bones, and the material / materials that contain textures.

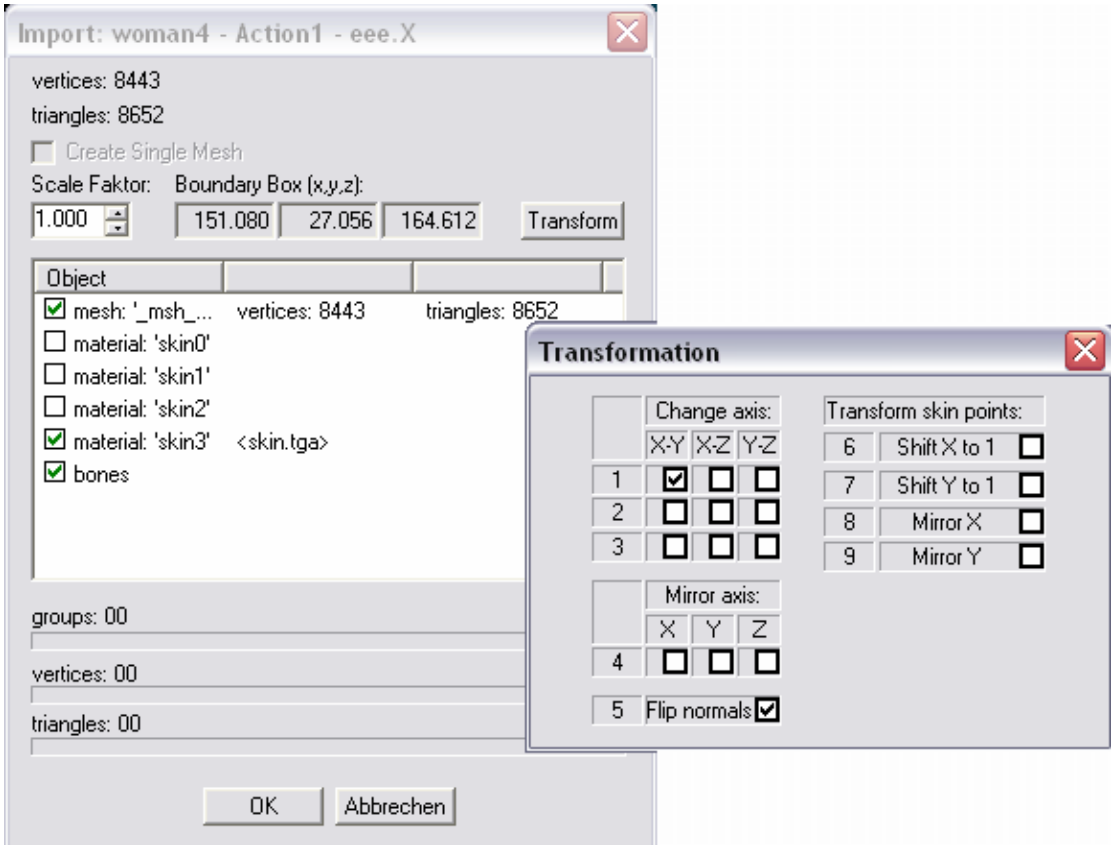


Figure 42: MED .X importer dialog and Transform sub-dialog

There should be no need for Scale Factor if the model has been created at the desired proportions in A:M. But, if we modeled in life-size and would want to use the 1 quant = 1 inch correspondence most 3dgs projects use, Scale Factor should be set to 0.4.

Set the Transformation options as shown above (Change axis / flip normals) so you have the model transformed to the Gamestudio default orientation. (facing right)

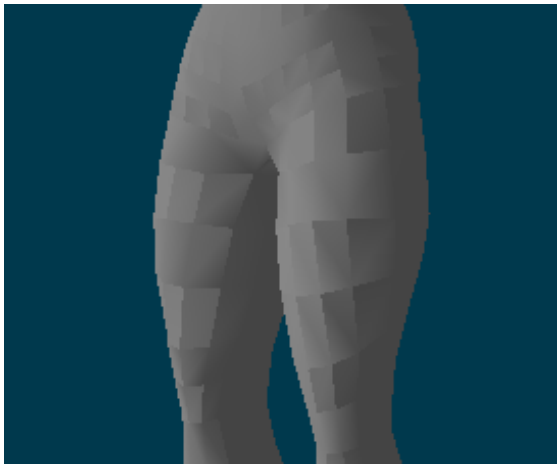
Cleaning up in MED

Geometry clean-up:

Exporting duplicate vertices in AMXTex is needed for accurate texture exporting, but it causes polygons to be split apart in MED.

"Broken" exported polygons

After vertex weld / merge



These Vertices can be weld with 3 clicks in MED

1. Select ‘Vertex Mode’ in the toolbar
2. Click on ‘Select All’
3. Click on the ‘Merge’ button

Figure 43: 3 clicks to 'weld' vertices

Texture / Material Clean-Up

Animation Master Groups turn into Material information in .X models. Usually we have to get rid of unwanted materials or material settings.
In MED, you can use the Skin Manager (Edit -> Manage Skins)

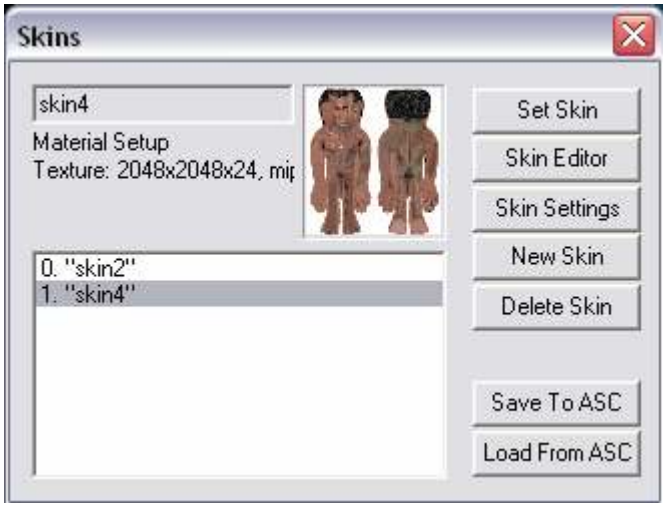


Figure 44: MED Skin Manager

Unwanted skins should be deleted in this dialog. Usually, all model parts are UVmapped in a single texture square, thus, we only keep the material that represents this texture. Delete unwanted textures and then click on Skin Settings.

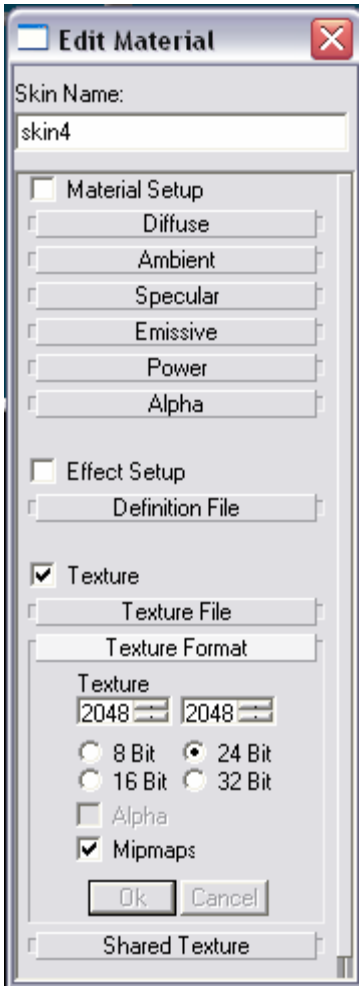


Figure 45: Skin Settings

Unless we are specifically using material properties, we want to turn off any Material Setup in Skin Settings (Uncheck the Material Setup checkbox).

Also, this dialog shows the Texture setup. Under Texture Format, we may convert our original Texture into another image depth.

Traditionally, A:M uses TGA for most textures. Since TGAs can have an Alpha channel, TGA textures are automatically set as 32 bit images with alpha when imported. This will lead to z-sorting rendering problems in Gamestudio.

Games don't handle this kind of textures very well for random models, unless for specific uses. You can convert your 32-bit image to 24-bit and no alpha without loss of color information. Also, it's very common to use 16-bit textures as they are smaller and save Graphics RAM. The drawback is that they have less color information and show coarse gradients. Still, 16-bit is the recommended format for most uses.

None of these settings affect the original texture file. Texture information is embedded in Gamestudio mdl7 files.

Animation clean-up

A small bug prevents Gamestudio to import the last frame of each animation sequence (scene). Provided that you added an extra animation frame (for non-cycle animations), or than you don't really need the last animation frame (for cycle-animations), you just should delete the last frame of each one of the scenes.

There are a couple ways to do this, but the easiest way is to use MED's Frame Manager (Edit-> Manage Frames).

[IMAGE: frame manager screen with deletion of last frames of scenes…]