Low-poly 3D Art Tips

April 20, 2010

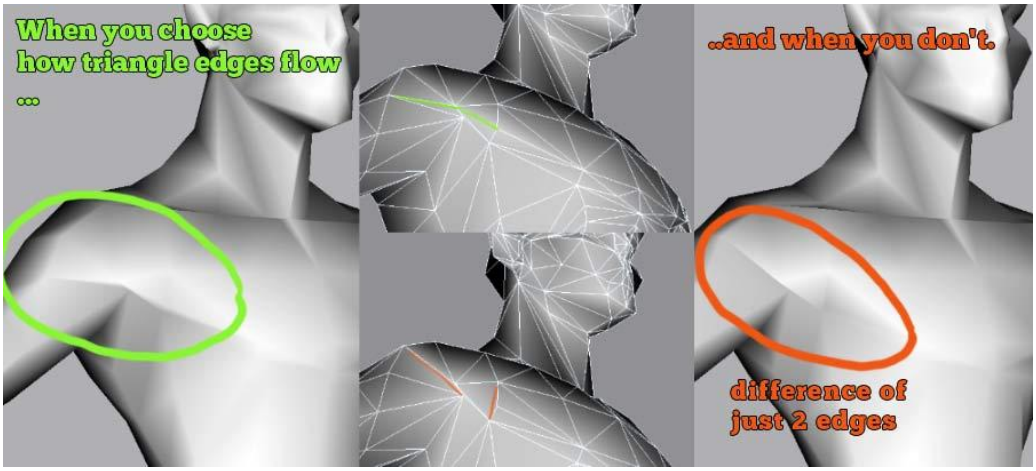
These are basic tips for optimizing and improving low-polygon work in the areas of modeling and texturing

Core of low-polygon work is to do a lot with little; Do something good with limited number of polygons and with small textures. Limitations are set by the game engine and platform the content is for – these days low-poly is usually for mobiles and other handheld devices. One very good way of getting the most out of your limited resources is doing stylized designs.

I recommend this: **Start by designing for low-poly**. One of the best examples of such styling is the look of World of Warcraft.

I learned low-poly working on the Ultima 6 Project. U6P uses Dungeon Siege(1) game engine and creates a large world filled with very low-poly models.

Make use of every triangle

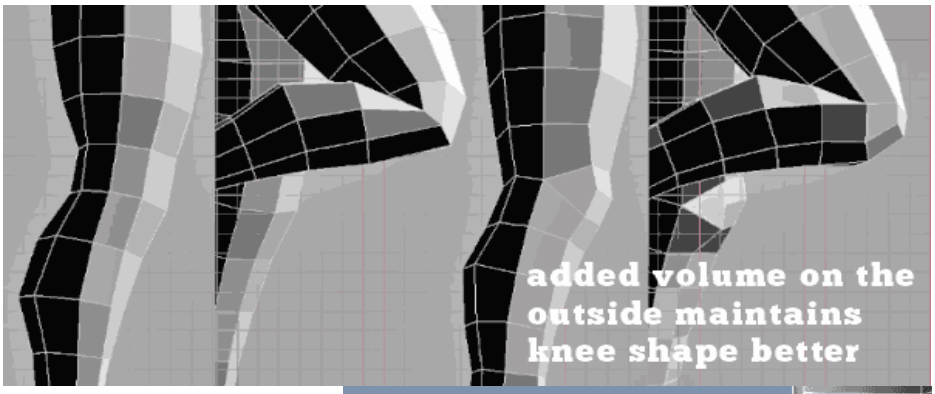


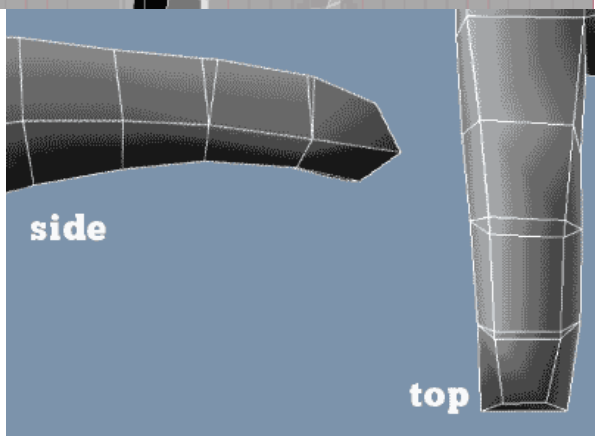
Sounds simple but is easy to overlook when people are used to modeling with quads or n-gons(more than 4 sided-polys).

Since every polygon is triangulated (divided to triangles) anyway when exported to game engine(or when rendered), you could just as well divide the polygons yourself. That gives you one more edge to define the shape with. The example shows how shape is created by placing edges dividing the quad polygons and what the result would be if the edges were misaligned – something you may get if you let the software triangulate for you.

Model volume on the outside of the joints

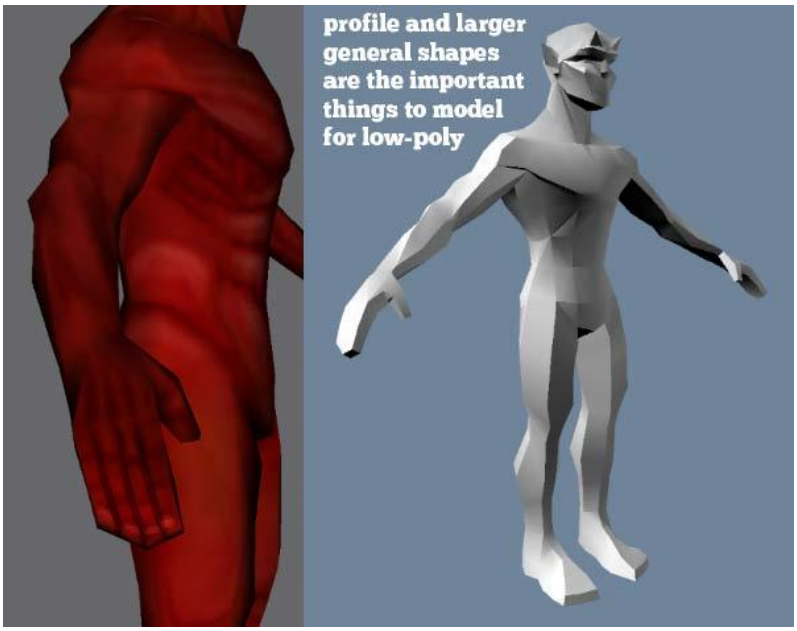
This way, when the limb bends, the outside preserves shape even when bent 'open'. The example shows setup you could use for knee or elbow and some others for fingers.





Focus in modeling the profile and the main shape landmarks

Polygons not used for better joint deformation or for defining general shape are extra – something you can do without. Create that extra detail with texture-map, instead.



Add shape with texture

by drawing some shadows&highlights into textures. But do it sparingly. Strong always present shadows or highlights look false.

Make holes with texture

Say you need a grate with lots of holes. Modeling them would mean a lot of polygons. Why not make plane and texture that with a transparent texture if your game engine supports it?

(Editor-note later: Very careful about this. Textures are also expensive, and need to be careful with this tradeoff.)

You could even have two planes, one see-through grate above and other below it with a well/whatever deeper place painted on it. Simplest solution is of course dark 'holes' painted in the main texture.

For more texture detail somewhere on your model make that part bigger in your UV-map

(Editor-note later: In other words give important parts higher texel-density.)

Sure this leaves less space for other parts, but some areas are more important than others (character face for one). Example has the gargoyle skin-texture with uvw-map overlayed.

Colour your textures by hand

Painting in shades of grey, black'n white, and then overlaying colour on might be easier, but if you instead both choose and paint the colours by hand, the end result is more vibrant and alive. Same applies to gradients. They tend to be too mechanical, too perfect. Paint the colour-shift yourself.

Use duplicate-parts in your character/object

Hands can often be mirror-copies of each other, same with legs. This is how you can save in UV-space and hence make all parts bigger in the UV-map and so more detailed. This is ever more true with objects like buildings where you can use same textures over and over. Clever and creative UV-worker can create many variations from just one texture-map.

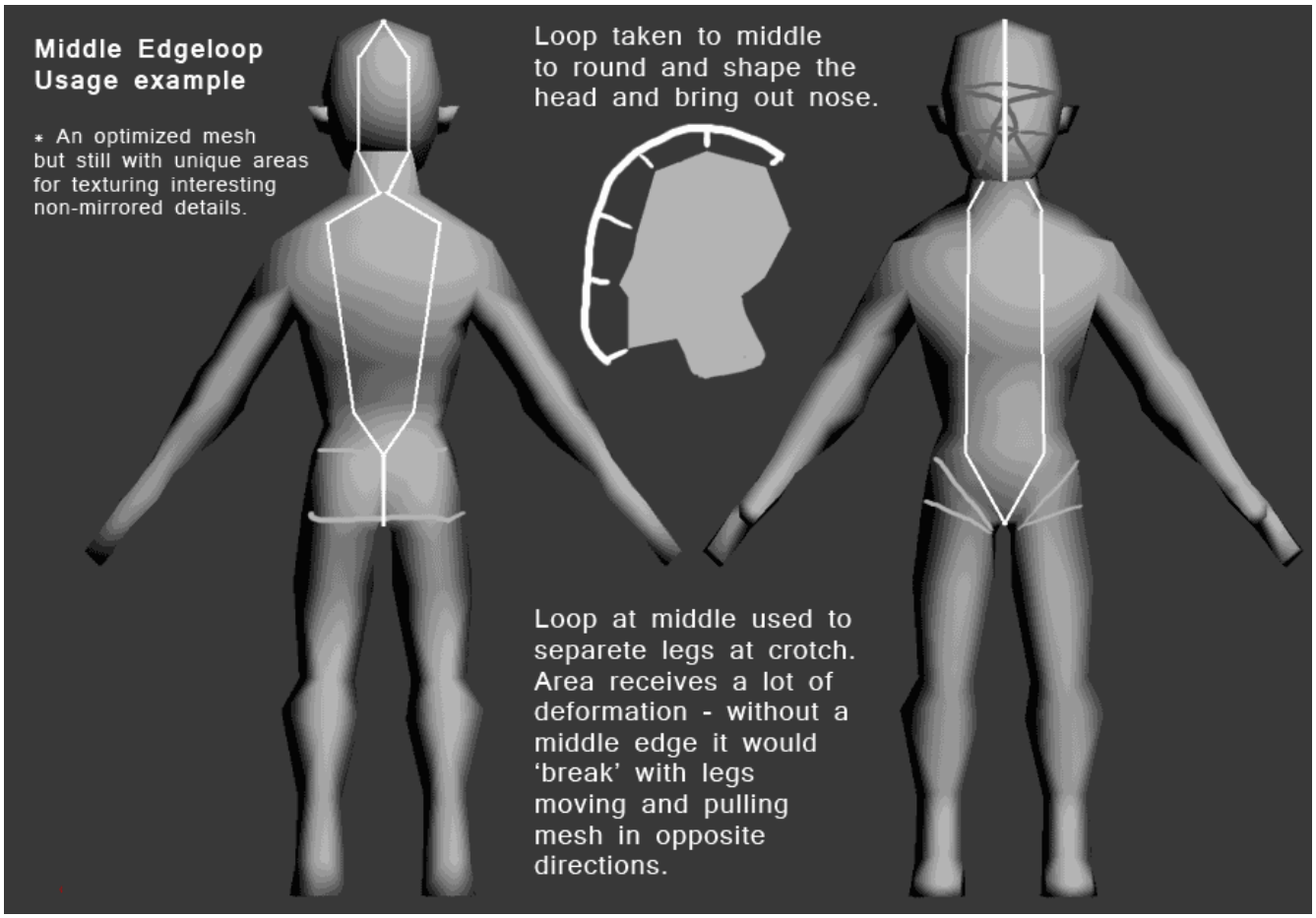
Low-poly Tips 2 – Game Art Asset Optimization

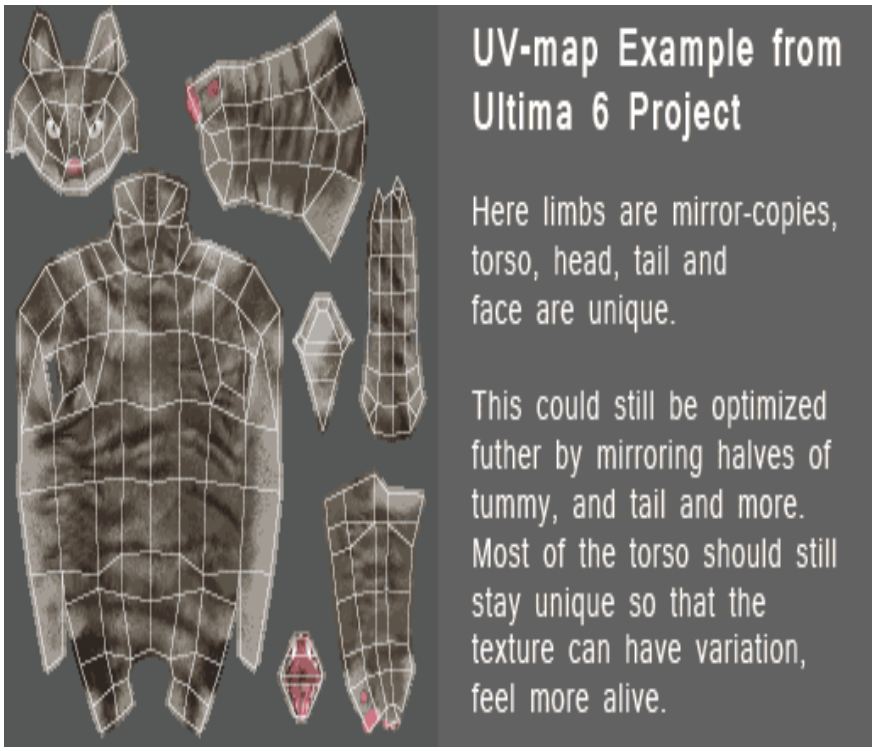
November 14, 2011

These modeling, uv-mapping and texturing tips apply to 3D art asset work for games and similiar media. While they are best matched with low-poly 3D, they are definitely not limited to it.

Middle edgeloop optimization & UV-mapping a character

It is common to model an edgeloop running around the middle of a character. It allows mirror copying the torso – you uv-map and texture only half and duplicate to get both halves with same detail. **However there are number of reasons why full middle loop and mirroring everything is often not the best choice**.

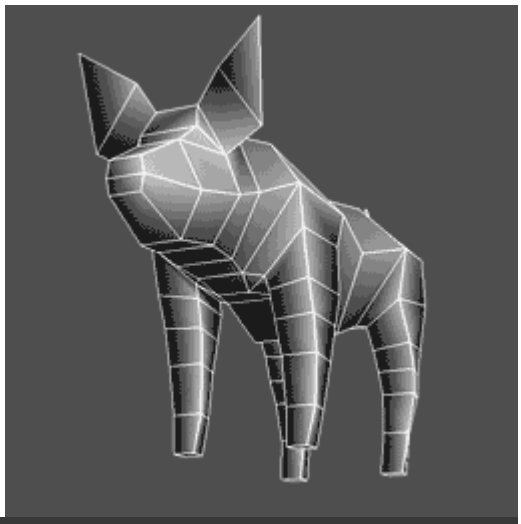




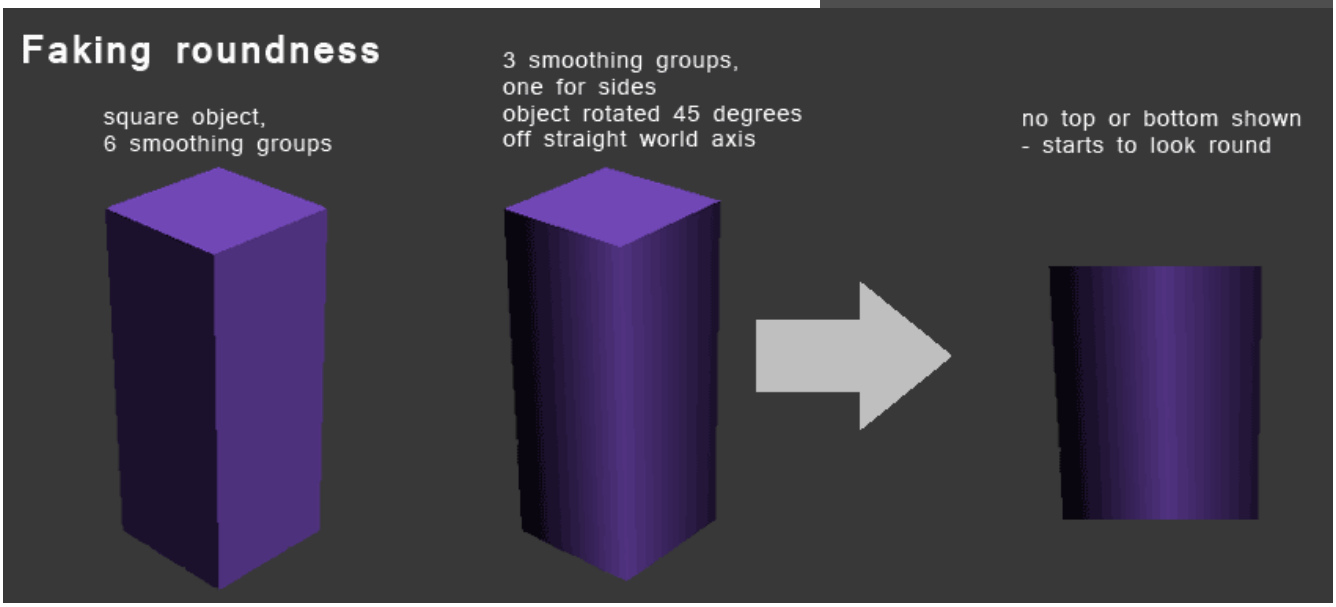
Mirror-uv-mapping everything on a character mirrored makes it look more generic. For visual interest you want variation in at least the texture if not the shape and for this you can't mirror everything. In a humanoid figure the places seen the most are where you want variation, usually top half of character torso, shoulders and face.

A middle cut running all through your character model means more polygons. There are places where you have to have it, namely the crotch/hips area for humanoids because this area receives lot of stretching – you need to separate the legs. But there are also many places where you don't need it. See the image for example of middle edgeloop use.

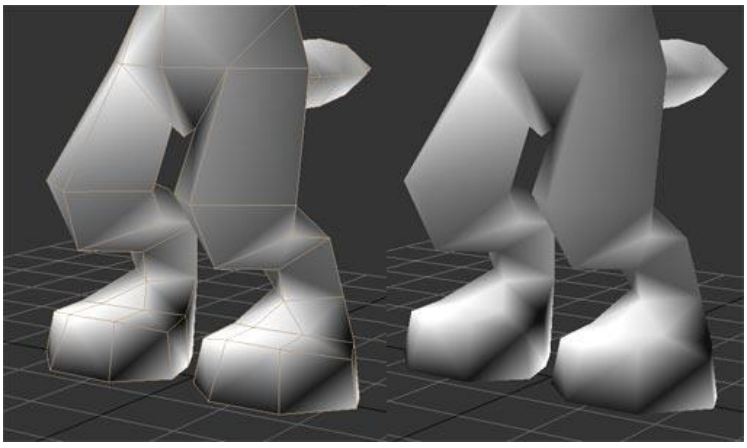
For four (or more)-legged characters like dogs you can often forgo the middle loop at hips, too. Sure the area will bend and break in animation, but if it doesn't show then does it matter?



Fake roundness with just 4 polygons – optimize asset polycount

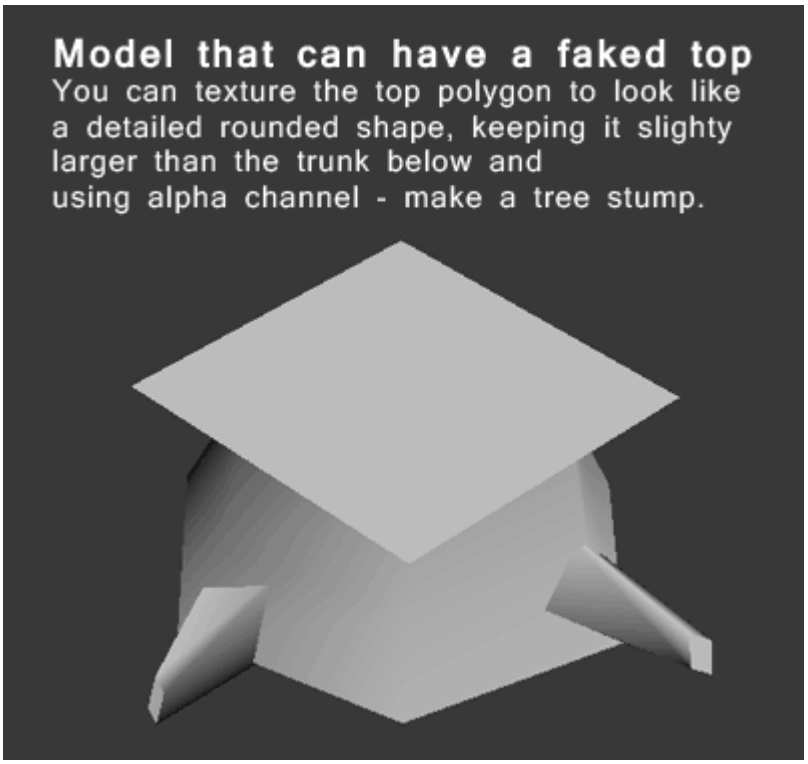


A square can be made to look rounded in game. The trick is to use one smoothing group and turn a square so that polygons are not aligned to world axis, rather angular to them. This places the corners closer to where round objects would be and away from where square objects corners were. That and the smoothing group fools your eye. It is mostly the smoothing group – I don't know the technicalities of this. Just that it works. See same tricks also with character legs.



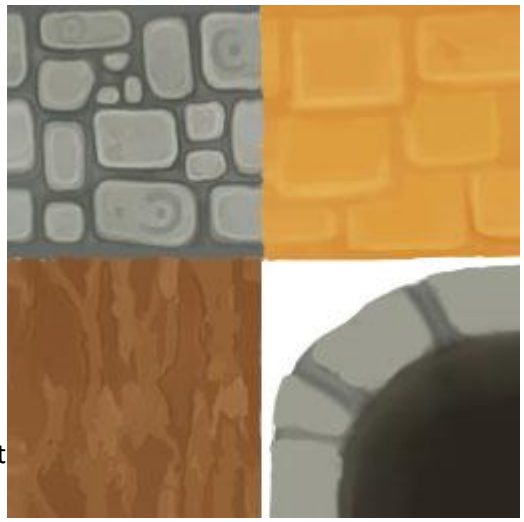
Of course this only applies to the sides, the 4 polygons we are talking about. Looking at the top and bottom the objects square nature shows, but when you hide them it is another story.

Fake complex shapes with bitmap and alpha channel – optimize polycount



Any object with a mostly flat top, especially shapes like barrel and similiar where top is equally proportioned or larger than parts below it, can have **a faked top: a single polygon and the shape of the top mapped on it with bitmap and alpha channel**. This can save numerous polygons. However the top with alpha does take space from your UV-map since it needs some size to have enough detail to not blur and reveal its faked nature. So judge for yourself which one is more important with your object: texture/uv-space or lower number of polygons.

Texturing with seamless textures – re-using textures

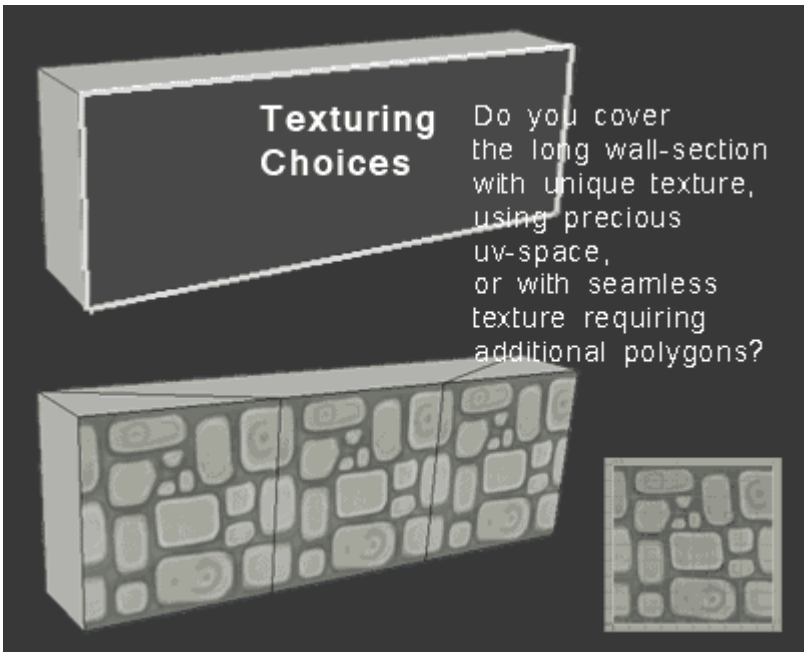


Re-using textures is a core part of low-poly work. Characters don't allow that too much, but props such as houses do. Say for a medieval building you might just have a texture with 1/4 stone, 1/4 wood, 1/4 roof, 1/4 window – see image used to texture a well, the idea is the same.

The trick is to place almost every polygon in your uv-map separately so that they grab the maximum texture area – AND also change polygon sizes, rotation and mirroring to add variation to the way it is displayed on your model.

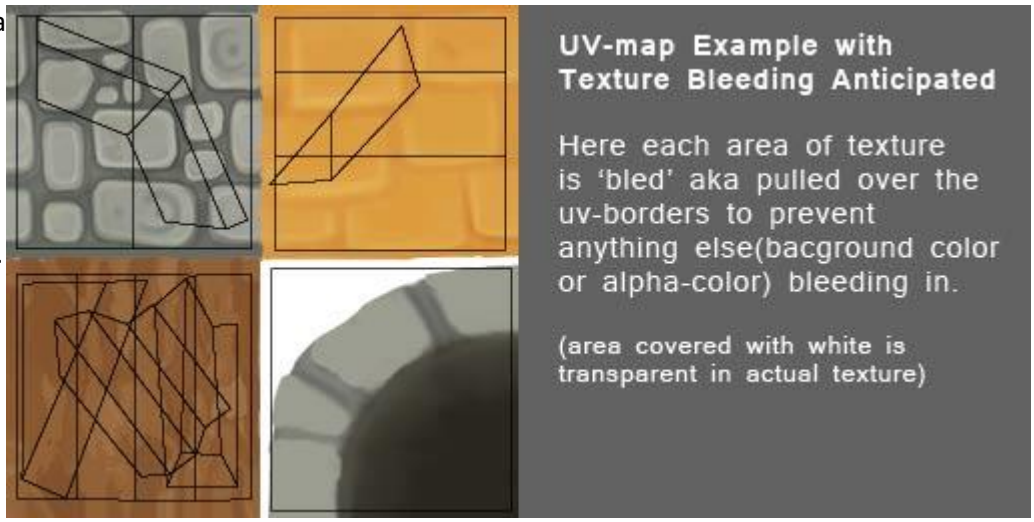
Unlike 'standard' uv-mapping, where you map first and texture after, for this you better do the reverse: Make the texture – lay out the different material areas(preferably each tileable). Think what you need and what shapes you need, like longer varied strips of material, and add those bits to your texture. Then uv-map polygon by polygon, or few at a time, to get all you can out of it.

Texturing by re-using textures does become a balancing act: Do you use more uv-space for one particular area or more polygons? Say you have a long continuous wall. To cover it all with a single unique texture would take a large amount of uv-space. On the other hand repeating one seamless texture over and over would require more polygons. So you weight the pros and cons and perhaps go middle way. Usually my take is that few polygons does less harm than needing to use larger textures or more textures.



Remember MipMapping and Antialiasing when texturing – stop texture bleed

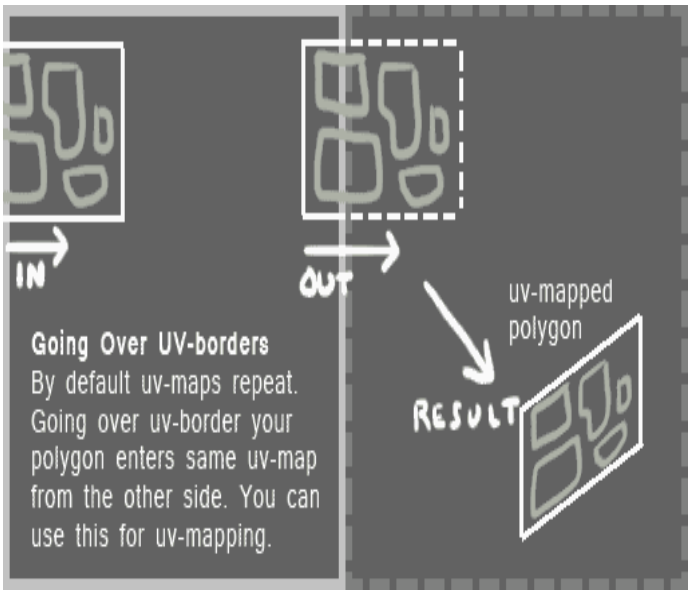
When the game creates a MipMap from your texture, or when the texture gets antialiased, it gets blurred. This is a problem at edges of the uv-island in your uv-map. Either the background color of your texture bleeds in or the alpha channel does(usually as black colour). As result the uv-edges become visible on your model in game.



To prevent texture bleeding problem, push the textures themselves well over the uv-seams. Then, when the blurring happens, you still have the correct colors at uv-seams.
http://en.wikipedia.org/wiki/Mipmapping

Acknowledge uv-area repeating – optimize texturing

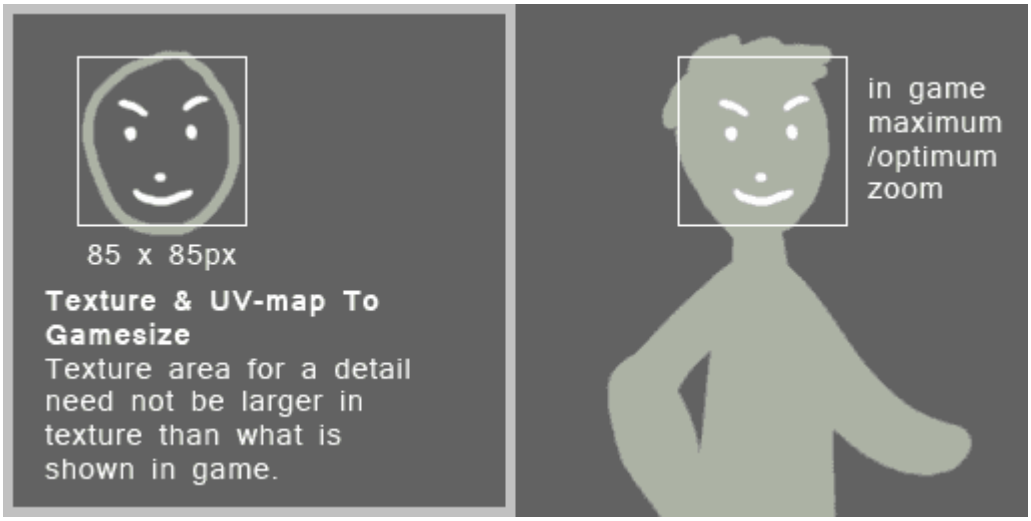
If a part of your uv-map goes over the uv-area, it will come out at the opposite end. This is not displayed visually in your program(not in Max or Modo at least), but knowing it you can use it to texture uv-parts that do not fit in your uv-space. Mind you this works only with seamless texture.



Have less seams in UV-map

Models that have their UV-map slipt to numerous parts count as having more vertexes as far as game engine is concerned – each split means more vertexes and so heavier to load. **To minimize vertex count you should have your uv-map as continuous as possible** – say a character skin could be one big open pelt like an animal skin. Of course uv-mapping and texturing poly-by-poly, like written above on seamless texturing, does the exact opposite.
Do note that going for less UV-seams is a fine-tuning type of optimizing – it is best used in addition to other tricks, where possible, and not to replace them.

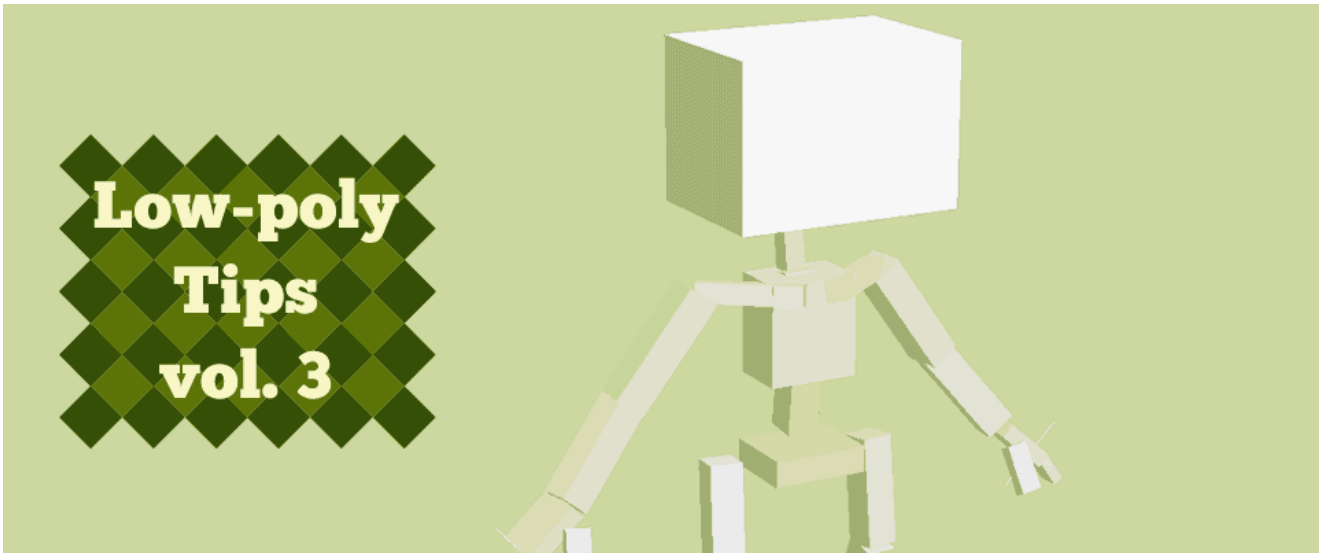
Make textures details to fit size displayed in-game – optimize textures



The size that the objects appear in game, be it because of optimum camera distance or whatever, defines maximum texture detail you need. Say you have a character face that is 85×85 pixels on screen in game. You need no more than that for it in the texture map. Of course if your game offers free camera, modifiable resolutions and such tools for player, things get more complicated. But even in free camera games there has to an optimum to aim for – what is the size of texture detail at camera distance where the game is Designed to be played at?

This ends second collection of art asset tips, especially useful when working with low-poly 3D assets. I hope some of these come handy in your projects.

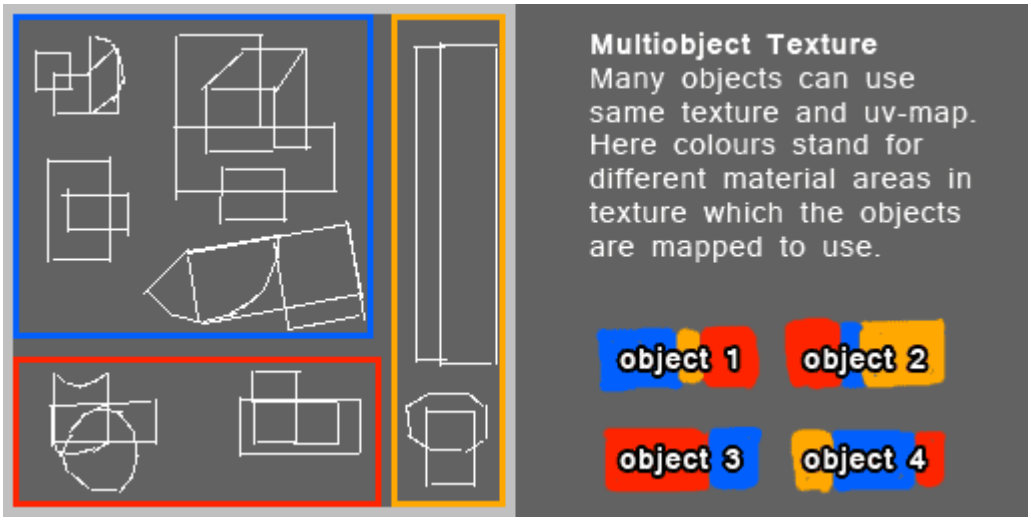
Low-poly Tips 3 – Game Art Asset optimization



These are 3D art asset modeling, rigging, uv-mapping and texturing tips. And not only for low-poly though it is where they are needed the most. See also other tip collections.

Minimize number of Draw Calls the Asset generates

Draw Calls are for game engine the number of separate objects, materials and textures that are loaded. The less draw calls the better the game can run. Here are some ways to lower the number:



Have each character as one single mesh. Characters that are made from pieces in-game cost in draw calls.

Combine separate static meshes to one. If you can have a collection of objects as one object, one file(the meshes can be unconnected), it is better than as several files. But don't combine a whole village to one object as the whole thing would get loaded to memory even

though you may not need it. This trick is best for moderate collection of objects, say all items inside a shop interior.

Use only one material and texture per object. Or even..

Have several objects all use same texture and material. This means each has the same uv-map but uses only a portion of the whole – uv-map collects all textures together. See picture. Even though not shown in picture (for clarity) the sections different objects use can well overlap.

Optimize character rig, use 2 rigs – one for animation and one for export

The less bones your character has the lighter it is to run. And less resources used for one character means more to use elsewhere – maybe even allowing more characters.

But very few bones makes animating difficult and prevents many motions. Of course we rather animate with the optimum amount – and with control objects as well to make work easier. Sure you can have control object in your game rig and just make sure not to export them to game, but having bones in a rig that you don't export, like between one bone and another? That is asking for trouble.

Solution is two rigs, one for animation and one for exporting to game. Game-rig is linked to follow animation rig – you animate only with animation-rig and export only with the game-rig.

Animation Rig is the rig you build first. It has the bones and control objects you want to animate with. The rig can even have details, like fingers, which you can animate and later decide to use or not (via the game rig). Build your animation rig and then consider what parts of it are essential for moving the character. Every bone in a character only supporting other bones and not really affecting the mesh itself is a bone the game character does not need. So, do you really need the neck-bone if head and chest bone playing together can offer the same result or close enough?

Game Rig is collection of helper objects (any type, also called nulls), one per every important part of character. The reason to use nulls instead of bones is that creating bones is intended to build hierarchies you don't need and should not have, here. Create these objects, then align and parent them to follow the relevant bones of Animation Rig. They should relocate to pivot-points of the animation-rig-bones. Then skin your character mesh to these helper objects(nulls). In the end you animate with Animation Rig and the Game Rig follows and deforms the mesh.