BLENDER MASTER CLASS

a hands-on guide to modeling, sculpting, materials, and rendering

DVD INSIDE

Ben Simonds

no starch press

5

MODELING THE DETAILS

In Chapter 4, we blocked in the basic elements of the projects, creating the block-in of the Jungle Temple and modeling base meshes for sculpting the Bat Creature and Spider Bot. In this chapter, you'll learn how to flesh out this framework to create finished models.

To create the final models, we need to transform our simple geometry using a mix of techniques. These techniques include using modifiers to add procedural details as well as modeling elements by hand. Our aim should be to end up with clean, well-modeled, detailed meshes, without overcomplicating things or introducing unhelpful geometry that could slow down renders or create artifacts.

Modeling details is great fun, but it can also be quite repetitive, so I won't exhaustively cover the process of making every part. Instead, I'll focus on key aspects of creating certain elements of projects that are the most interesting or tricky and leave the rest to your imagination. We'll begin with some discussion of topology, discuss what constitutes a "good" mesh, and then move on to the actual modeling.

Topology

Topology describes the way that the edges and faces of a mesh connect and flow across its surface. We covered the basics of topology when creating the base meshes for sculpting the Bat Creature and Spider Bot—namely, creating even loops of vertices around the arms and legs and avoiding triangular faces. Now let's talk about why we do things this way.

There are many ways to create meshes that have the same basic shapes but use very different configurations of faces in their construction, as you can see in Figure 5-1.

While the shapes in Figure 5-1 are roughly the same, the middle mesh is the most useful because its geometry flows with the form of the face, creating loops around the eyes and the mouth and running neatly down the neck and over the head. Also, it describes the forms of the head just as well as (or better than) the other meshes, while using fewer faces.

The flowing characteristics of this mesh are also important for animation because they allow the mesh to deform easily and smoothly. For example,

closing the eyes or opening the mouth won't stretch edges awkwardly or cause parts of the mesh to intersect unpleasantly. This kind of loop-based topology also helps when creating further variations on the shape, and it makes it easier to place UV seams and to UV unwrap the mesh without too much stretching (see Chapter 8 for more on unwrapping).

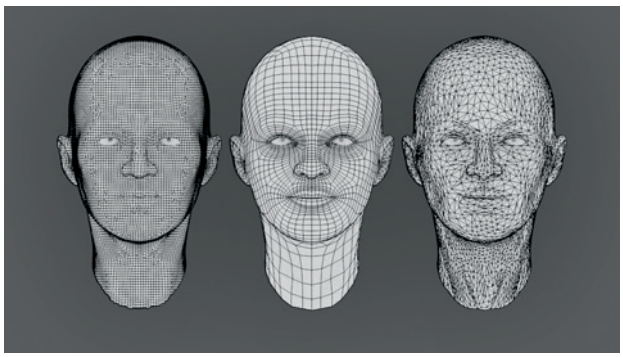


Figure 5-1: The same head shape with three very different meshes

Another reason that the middle mesh in Figure 5-1 is the better choice is that its topology is the best suited for use with the *Subdivision Surface (Subsurf) modifier*. The Subdivision Surface modifier, which we covered in Chapter 4, is used to subdivide and smooth a mesh. The algorithm used by the Subdivision Surface modifier, Catmull-Clark subdivision, works best when given a mesh constructed like this one. When the Subdivision Surface modifier is used with a mesh containing a lot of triangles or long, oddly shaped faces, it can give poor results, but when given well-constructed, flowing topology, it produces very predictable, smooth forms.

What Is Good Topology?

Good topology for animation is usually good for subdivision and vice versa. But what constitutes good topology? While there are no absolute rules, there are a few important principles. It's a mix of art and science.

Avoid triangles and *n*-gons where possible. This is the big one. While triangles are fine in a static mesh that you don't intend to subdivide or in a low-poly object for a game, if you plan to subdivide your mesh, use as few triangles as possible because triangles don't subdivide as well as quads. Equally, *n*-gons are converted to triangles before being subdivided, resulting in the same kinds of problems.

Avoid poles with lots of edges. A *pole* is a vertex where three, five, or more edges meet—that is, a point in a mesh that deviates from a grid-like structure. Like triangles, poles can create artifacts when subdividing a mesh. Poles with three or five edges aren't so bad—indeed, it's just about impossible to create anything but toroids and grids without creating a few poles—but poles with six or more edges subdivide poorly.

Create loops around important forms. This allows you to easily select, deform, and animate your meshes, and it also ensures they will subdivide cleanly. For example, in Figure 5-1, the use of edge loops that flow around the eyes makes it easier to adjust their shape.

Align edges with the form. If your object is roughly cylindrical, the edges of the mesh should flow around its circumference and along its length. If your object is roughly cuboidal, create it by starting from a cube and adding loop cuts. In general, try to create a mesh structure that goes with the "grain" of the shape you are trying to create, as shown in Figure 5-2.

Dealing with Difficult Topology

The rules listed above are simple, but you may run into trouble following them from time to time, especially when trying to eliminate triangles and poles from your models. Here are some tips for dealing with difficult topology:

Plan ahead. Most topology woes can be sidestepped simply by planning ahead. That's why, for example, we made sure there were eight vertices in the loops around the arms and legs when creating the base mesh for the Bat Creature: It made joining the hands easy, as there were no surplus edges to join together when it came to bridging the gap. Powers of 2 (8, 16, or 32) are often a good way to think about this, but regardless, try to keep to even numbers when creating edge loops. If you are new to 3D modeling, it can be helpful to sketch your desired mesh over a photo or your concept art, either in GIMP or on paper, as shown in Figure 5-3.

Two tris make a quad. You can join two adjacent triangles to make a quad, killing two birds with one stone. To convert multiple triangles into quads automatically, select your mesh and press ALT-J to turn suitable pairs of triangles into quads.

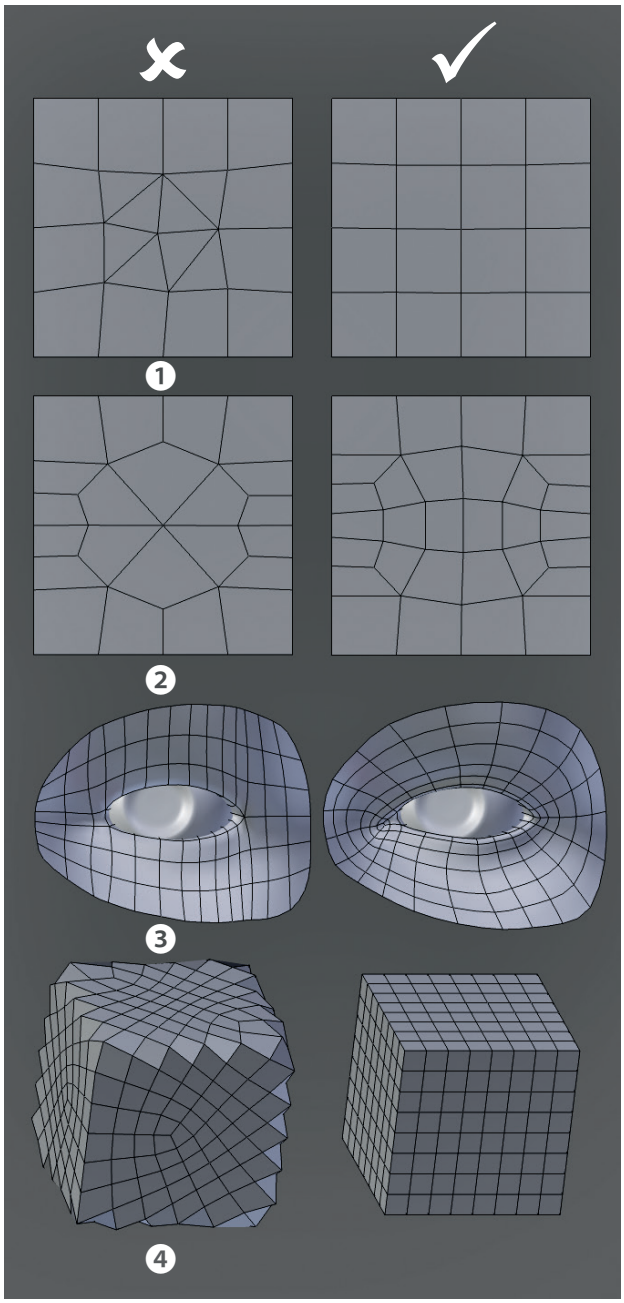


Figure 5-2: Topology dos and don'ts: Avoid triangles ❶, avoid poles ❷, create loops around important forms ❸, and align edges with the forms, not against them ❹.

Rotate edges to move triangles. To rotate or "spin" an edge, select it and press CTRL-E ▸ **Rotate Edge CW/CCW** (clockwise/counterclockwise). This rearranges the faces around that edge, allowing you to move triangles. You can combine this trick with the joining adjacent triangles trick above: By spinning edges to bring two triangles together, you can eliminate them, as shown in Figure 5-4.

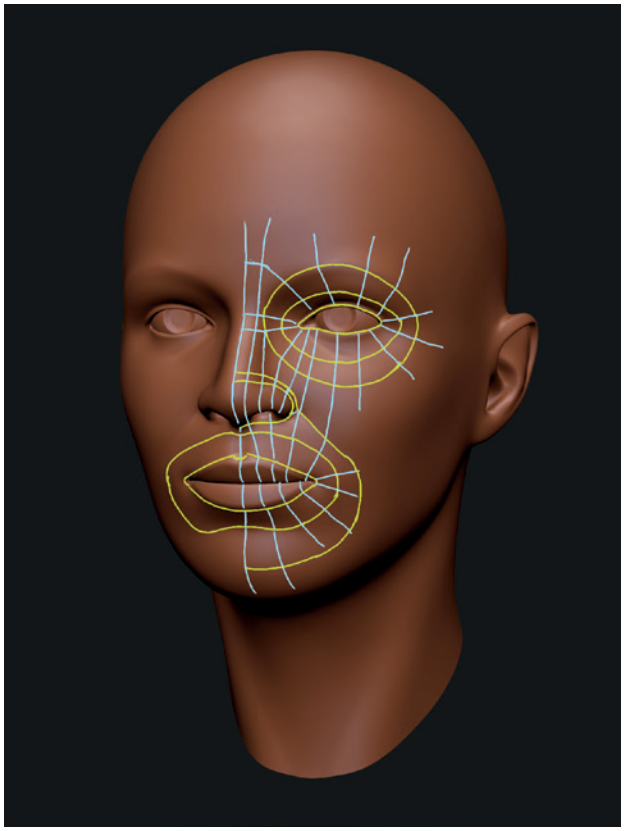


Figure 5-3: Sketching your topology beforehand can help you avoid difficulties.

Add an edge loop. Adding a loop cut (CTRL-R) that ends on a triangle will turn that triangle into a quad (or two triangles, which you can merge into a quad). If the new edge loop terminates at an open edge, you've eliminated your triangle. If your mesh is closed, it might just move the triangle to the other end of the edge loop, and, if you have triangles at both ends, you can take out two at a time (see Figure 5-5).

Split a pole in two. A pole with six edges can easily be split into two fives by adding a face loop between the two halves. Add further faces for even cleaner topology, as shown in Figure 5-6.

Cut, dissolve, and join. The cut tool (**K**) allows you to arbitrarily cut edges and faces to get the topology you want. You can combine this tool with the Dissolve operator (X ▸ Dissolve)

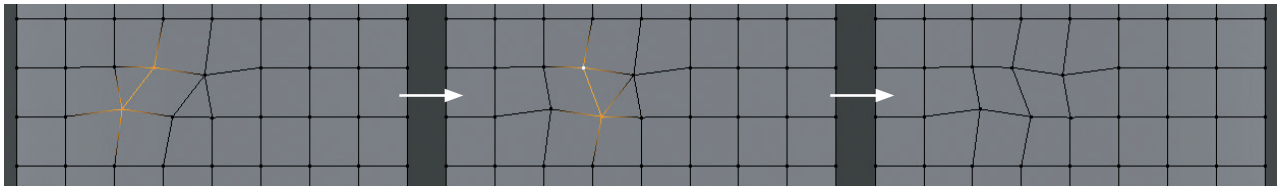


Figure 5-4: Rotating edges to bring two triangles together allows you to eliminate them.

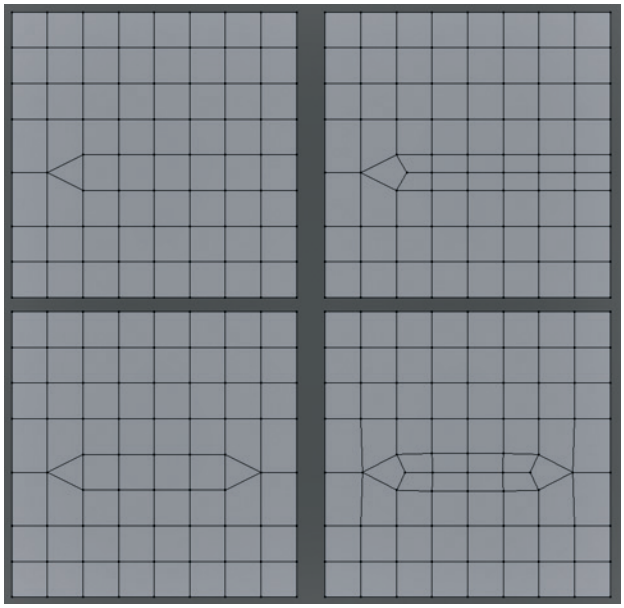


Figure 5-5: Adding edge loops with the Loop Cut tool (CTRL-R) can get rid of triangles. If you add one between two triangles, you can eliminate them both in one go. Alternatively, you could delete one of the edge loops already present with similar effect.

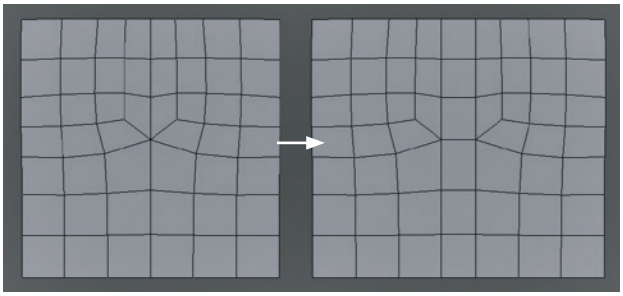


Figure 5-6: Adding faces to remove poles. Adding one face loop across six poles reduces it to two five poles, which will subdivide much more cleanly.

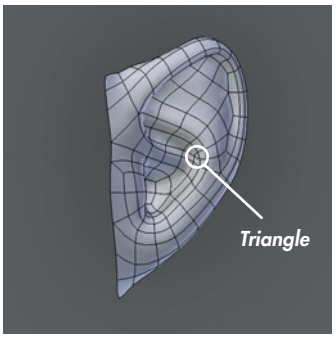


Figure 5-7: This triangle hidden in the corner of an ear isn't likely to cause much trouble.

to get rid of vertices' edges without deleting the faces they are part of. Then use the Join Edges operator to connect two vertices that are already part of a face but do not have an edge connecting them. These tools are great for arbitrarily rearranging difficult topology.

If you can't get rid of it, hide it. If you really can't get rid of a triangle, hide it where it won't cause trouble or create subdivision artifacts, such as inside an ear or nostril or someplace really flat that doesn't need to deform, as shown in Figure 5-7.

When in doubt, start over. If you managed to make something once, chances are you can do it again and get it right. It might take some extra time, but it's usually worth it.

Modeling the Details of the Jungle Temple

The Jungle Temple scene was already blocked in, so next it needed to be refined to make it more final. This process required me to think about the shapes I wanted to create and to model more complex meshes with the shapes I wanted in the final renders.

Walls

For the main walls of the Jungle Temple, I first laid out cubes to form the stone blocks of the wall (see Figure 5-8). Beginning with the bottom row, I added each one by hand and modified its length to give some variation. Next, I built up the higher layers by duplicating and scaling the cubes. To add further variety, I selected blocks at random, moved them in or out from the wall a bit, and rotated them slightly to make the surface of the wall somewhat more uneven. Leaving the basic walls from the blocking-in stage behind the new blocks provided a filler for the gaps between the blocks.

To add a beveled edge to the blocks (see Figure 5-8), I subdivided them a couple of times (select all [**A**] in Edit mode, then W▸Subdivide) and then added a Bevel modifier and set the "limit" method to Angle. The limit restricts the beveling to edges between faces at a sharp angle, and setting the angle to about 45° gives a nicely rounded bevel that is heavier on the corners of the blocks than at the edges. The sides are left alone.

*When applied, the Bevel modifier can create errors that will turn your geometry into triangles and create a lot of duplicate vertices. To fix this, apply the modifier only once you're finished modeling. Then, in Edit mode, select everything (**A**) and use the Remove Doubles (W▸Remove Doubles) operator to eliminate duplicated vertices. Next, use the Triangles to Quads operator (ALT-J) to return to a cleaner mesh without so many triangles. You can also bevel individual edges and vertices in Edit mode, using the Bevel operator (W▸Bevel).*

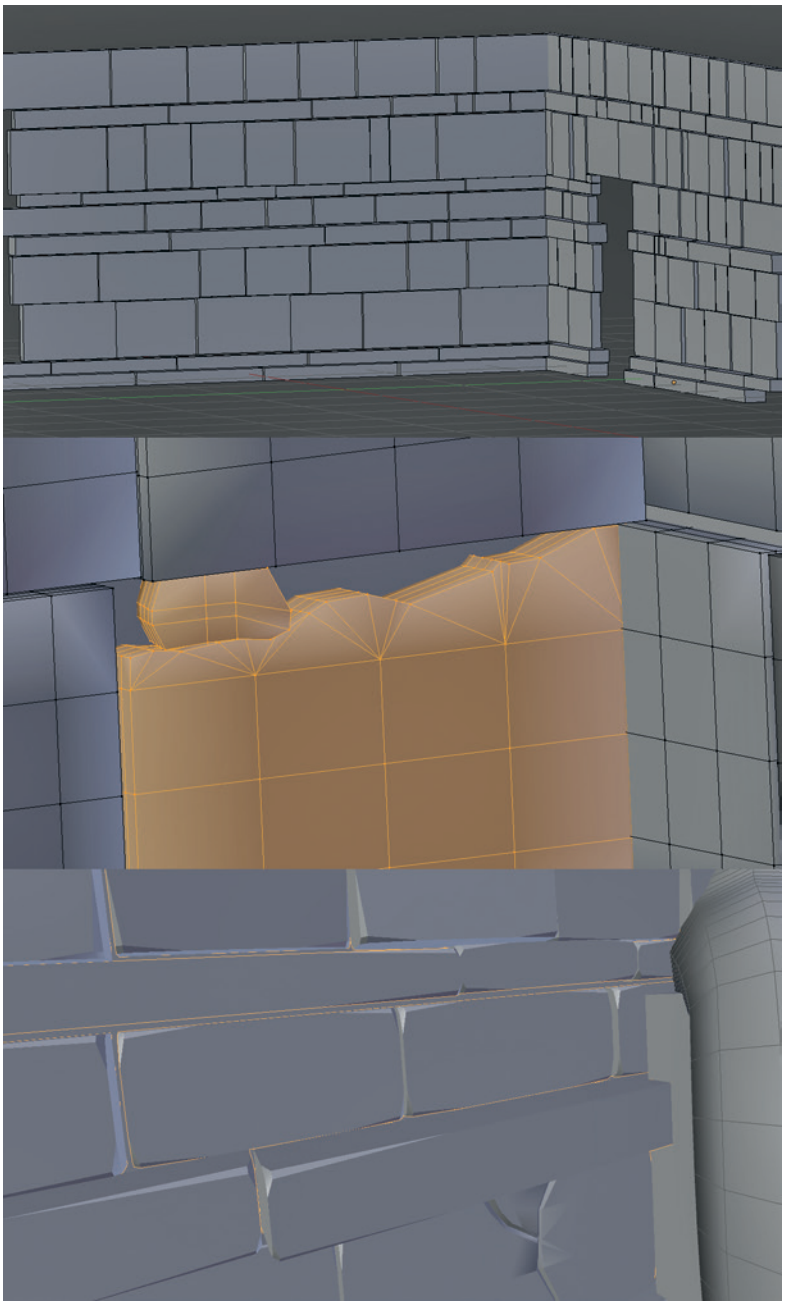


Figure 5-8: Creating the stone blocks for the walls. First, I blocked in the walls with simple cubes, which I scaled and moved to build up the wall. Next, I damaged the walls a bit by adding some basic subdivisions and roughening some edges. Finally, I beveled the edges of the blocks using a Bevel modifier limited by angle.

For the block details, I subdivided some of the blocks and then added extra features, such as a crack down the middle, a chunk out of a corner, and a split. I added loop cuts or subdivided specific parts and moved vertices around to create cracks, dents, and chips. Because the mesh won't be subdivided or deformed significantly, there's no need to avoid triangles here; they won't cause problems.

To prevent the blocks from looking faceted, I set their shading mode to Smooth and then added an Edge Split modifier to split the mesh at certain edges in order to produce separate surfaces (see Figure 5-9). The Edge Split modifier breaks the mesh into separate pieces so that when shaded smooth or when further modifiers are applied, the edge between the pieces is preserved. You can set Edge Split to split the mesh either along edges tagged as Sharp in Edit mode (CTRL-E ▸ Mark Sharp) or along edges with sharp enough angles between their faces. Using the angles only with a setting of 30° resulted in nice-looking blocks.

I created the other incidental blocks and paving slabs in the same way as the walls, using the initial block-in cubes as a guide for placement and then deleting the old geometry once the new blocks were placed. The final blocks are shown in Figure 5-10.

Statues

I modeled the statues in the corners by the door of the Jungle Temple using fairly basic building blocks and my concept art as a guide. Each part began with a simple primitive—usually a cube or cylinder—transformed, subdivided, and extruded to create what I need.

As shown in Figure 5-11, each part is fairly simple. To add beveled edges, I used the same method that I used for the wall and floor blocks. Adding some loop cuts around the ends of some pieces (for example, to the "legs" and the ends of the arms) allowed these edges to retain their square shape and sharper corners when beveled, without too much subdivision.

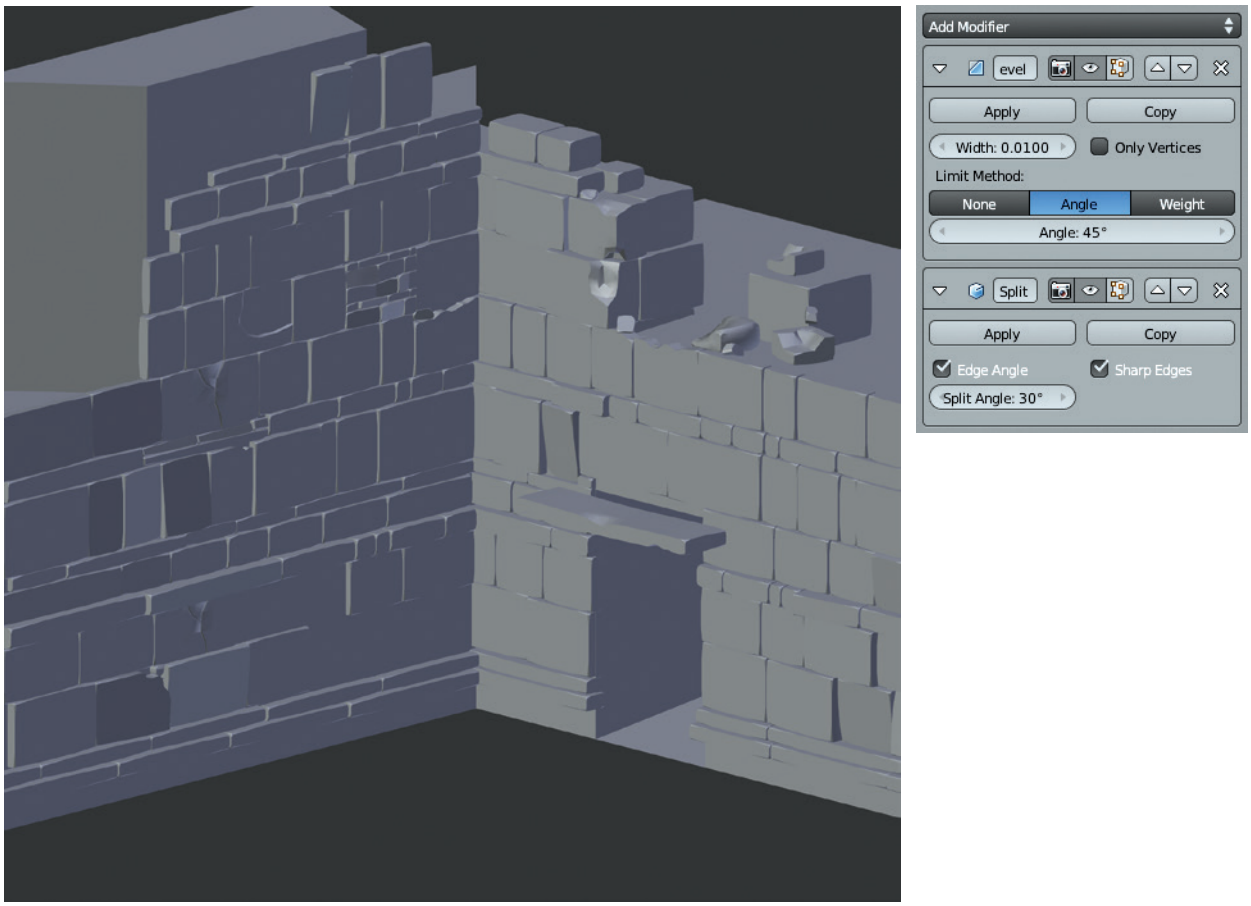


Figure 5-9: The final walls, with a Bevel and an Edge Split modifier to give them beveled edges and flat sides

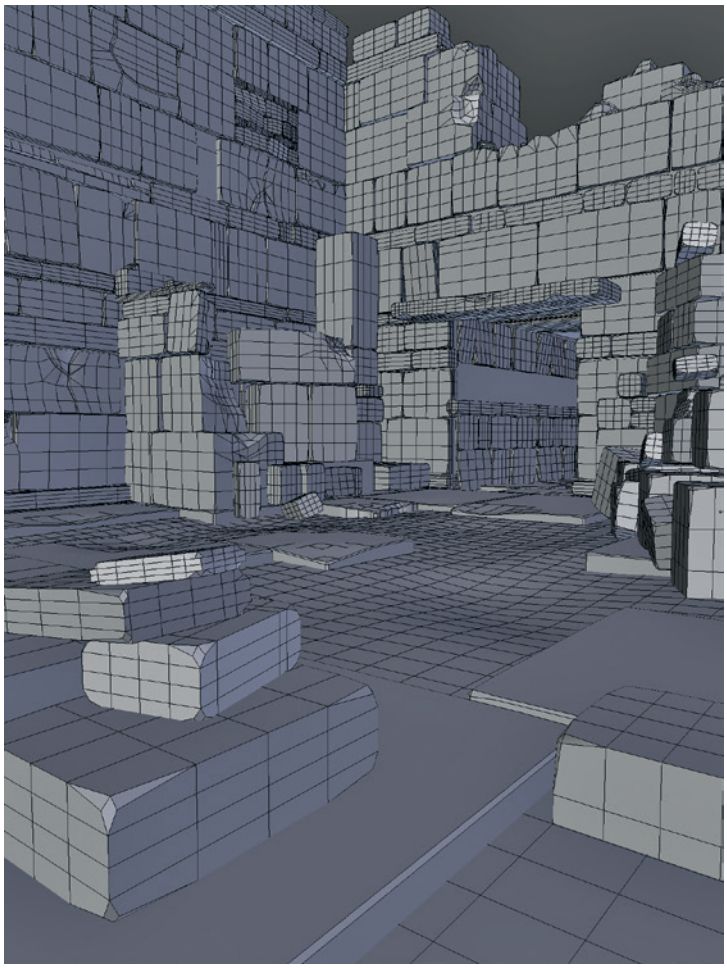


Figure 5-10: The rest of the stone blocks in the scene were modeled in the same way as the walls.

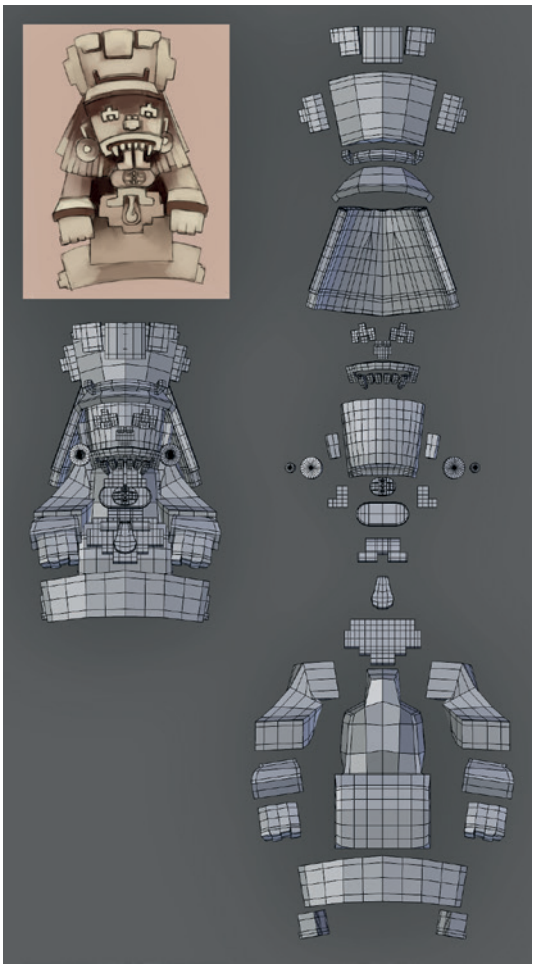


Figure 5-11: The statue model exploded into its constituent parts. Most are derived from simple cubes or cylinders.

Stone Carvings

For the stone glyphs, I used my concept art as a single orthographic reference, loading it as a background image, as discussed in Chapter 3. Then, beginning with a plane, I traced each piece of the designs, sticking primarily to quads where possible. Next, I extruded the whole design downward to give it some thickness and deleted the new faces afterward to leave just the sides and front of the design. By placing edges along the forms of the design's interior elements, I could move the grooves in the design downward to create the details.

To clean up the design, I used creasing (see Figure 5-12). By adding a Subsurf modifier and tagging edges as creased, you can create smooth objects with sharp creases along the tagged edges. Creasing allowed me to add tight creases to the model without using more polygons than necessary. Adding an Edge Split modifier after the Subsurf modifier then gave a nice smooth mesh with sharp transitions at the creased edges.

With the carvings complete, I moved on to placing them in my scene by replacing some of the rows of stone blocks in the walls with rows of the glyphs. To do this, I lined up all six glyphs in a row and then used an Array modifier to repeat the design to fill the length of the wall (see Figure 5-13).

Tagging Edges

The edges of a mesh can be tagged or marked in a variety of ways, each of which tells different Blender operators and modifiers how to perform operations

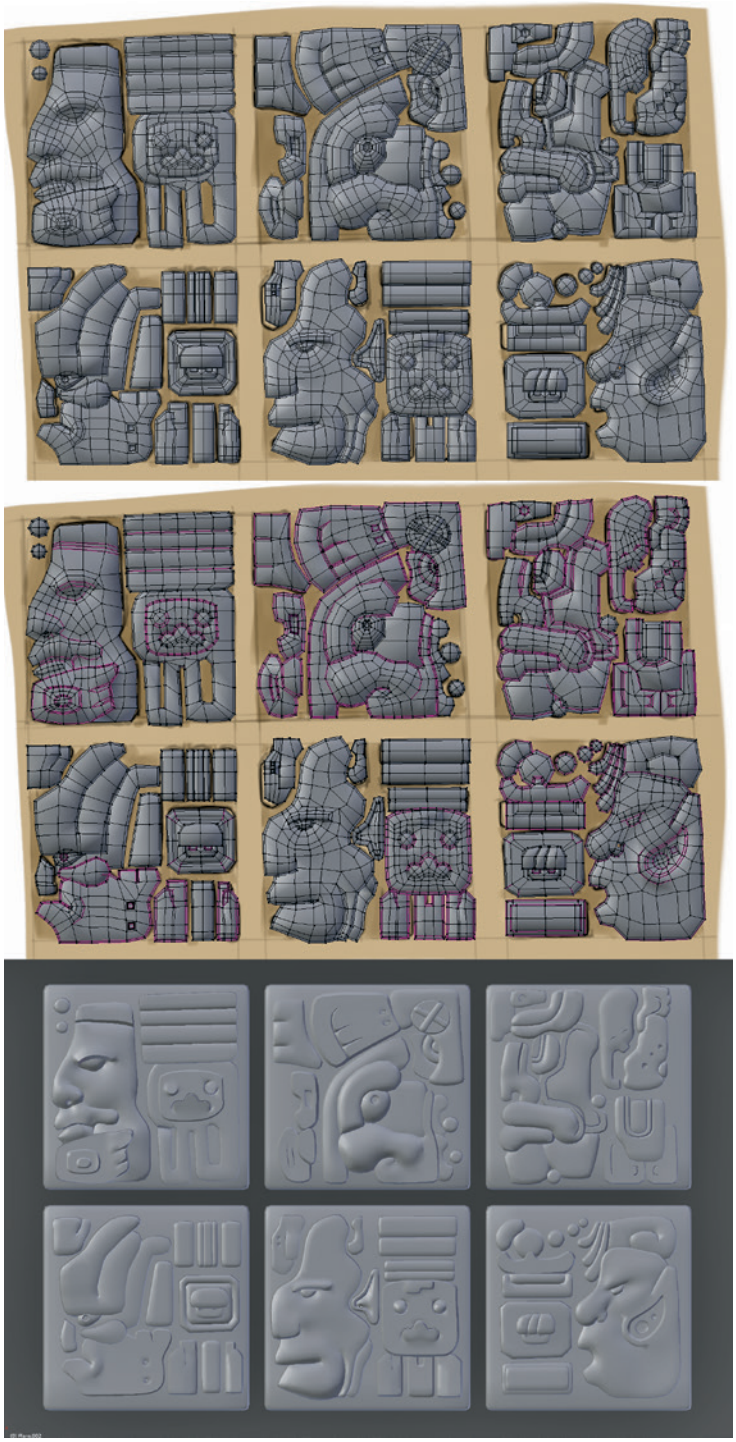


Figure 5-12: Creating the stone carvings. I first blocked out the carvings as individual pieces over the concept art, using primarily quads. Next, I tagged some edges as creased (purple) to give sharp edges when subdivided. Finally, I added Subsurf and Edge Split modifiers to give smooth carvings with sharp edges where the edges had been tagged. The result uses fewer polygons and simpler topology than if I had used support loops to produce sharp edges.

on the mesh. The shortcut for Edge operators, including Tagging, in Edit mode is CTRL-E. Edges can be marked as Sharp, which allows operators like Bevel and Edge Split to work only on these edges. They can also be given a *crease* value, either from the CTRL-E menu or with SHIFT-E, which tells the Subsurf modifier not to smooth these edges when doing subdivision, resulting in nice sharp edges.

Sharp Edges and Subdivision

Creasing is one way to achieve a sharp crease in a subdivided mesh; another way is with *support loops*. Support loops work by placing two or more edge loops close together at the edge of a form. When the mesh is subdivided, the new geometry can't be smoothed out as much because the extra geometry defines the corner more tightly (see Figure 5-14).

Both methods have their place. Support loops are the better choice when you're trying to produce nicely beveled edges and need fine control over precisely how your forms look. However, if your goal is to produce very sharp creases or you are working with simpler models with less extra topology, creasing is preferable. You can always mix and match both methods as the situation demands.

Support loops are also useful for ensuring that objects subdivide to give the shapes you want. For example, Figure 5-15 shows a cube with a Subsurf modifier applied and its wireframe visible. With no support loops, the modifier turns it into a sphere. With extra support loops running around the middle of the faces, the object more closely resembles a cube, and shifting these support loops toward the edges of the cube makes the corners sharper. This is useful when modeling all sorts of surfaces.

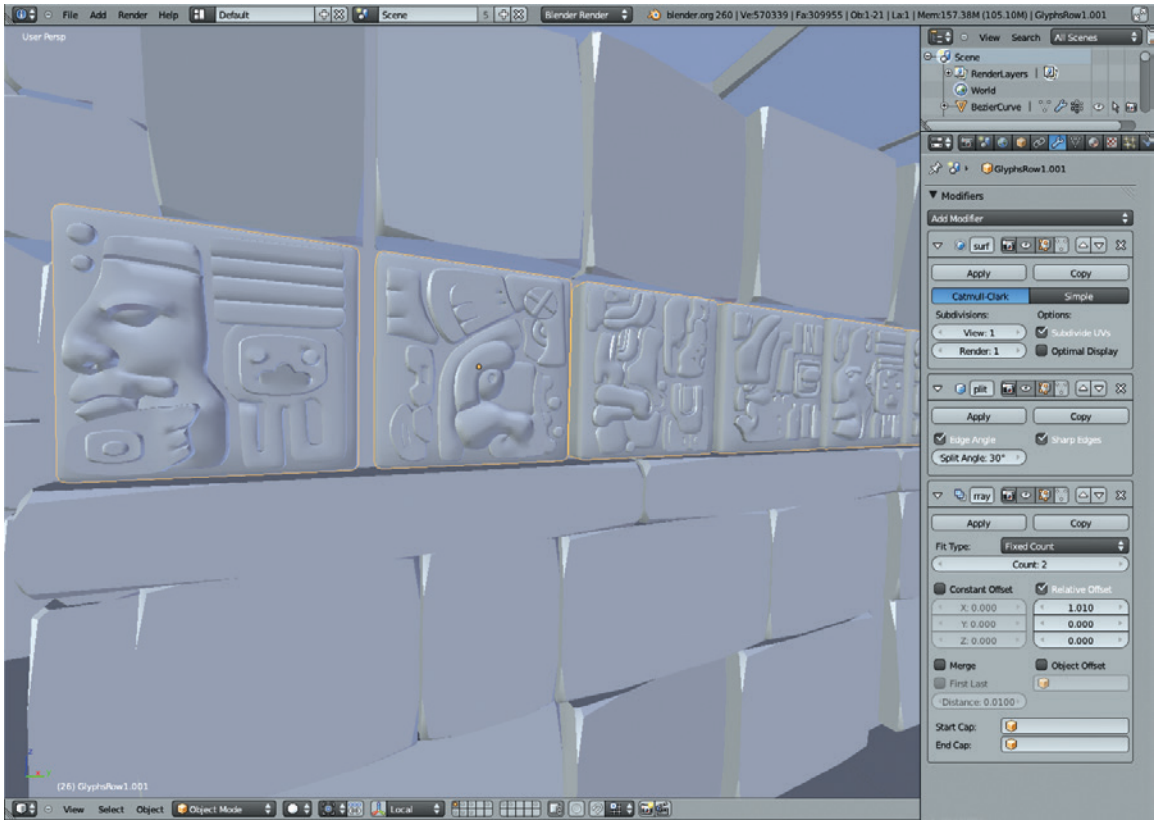


Figure 5-13: Using the Array modifier to repeat the stone carvings multiple times

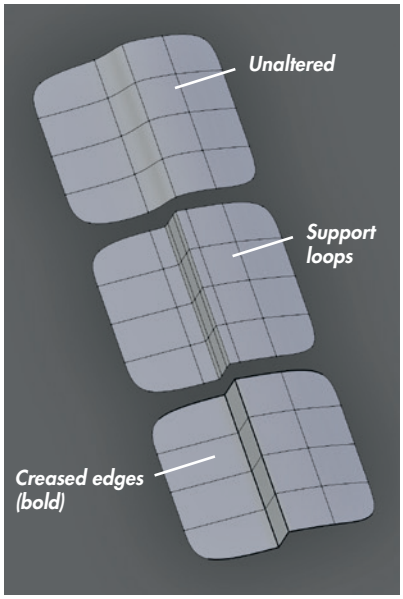


Figure 5-14: Two methods of getting sharp edges when working with a subdivision surface. Top: Unaltered mesh with a gentle slope and a Subdivision Surface modifier applied. Middle: Support loops added to give sharp edges. Bottom: Edges creased to give sharp edges (without extra geometry).

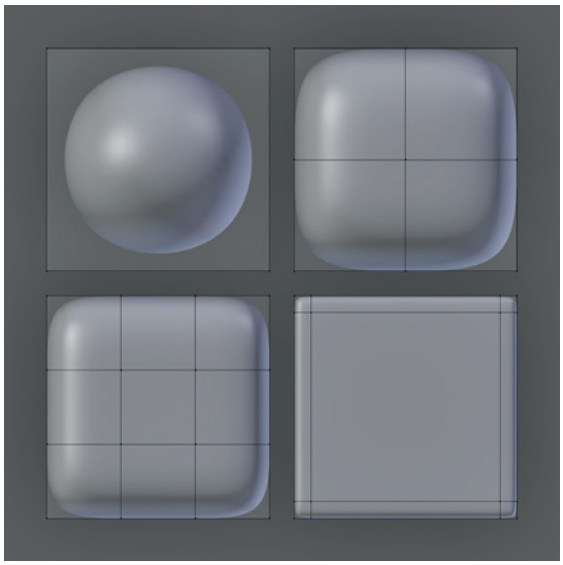


Figure 5-15: Clockwise from top left: A cube with zero loops, one loop, two widely spaced loops, and two evenly spaced loops running around the middle of each side. As the edge loops get closer to the edge, the corners become more sharply defined.

Plants

The plants in this scene are simple. To create them, I began by modeling a few varieties of leaves from planes; I scaled and subdivided the planes into leafy shapes (see Figure 5-16). Next, I started duplicating these different leaf objects and placing them around the scene. By creating linked duplicates with ALT-D, you can create multiple copies of the same mesh that all update together when you change one duplicate, which makes creating UVs and textures much easier because you only have to do so once for each type of plant. By scaling and rotating these duplicates in Object mode and placing them around the scene, you can give the impression of a lot of variation without having to create a lot of different meshes (see Figure 5-17).

There are two ways to duplicate a mesh in Object mode. One way is to create a simple copy, which then becomes a unique object (SHIFT-D); the other is to duplicate a linked copy (ALT-D), which retains the same mesh data and materials as the original and updates along with it. You can still apply different modifiers to a linked duplicate and move, scale, and rotate it independently in Object mode, but its mesh data and materials, as well as other data, will remain linked with the original object; if you edit one, the changes are applied to both.

Both methods are extremely useful for different tasks. Basically, you should use simple duplicates in the following situations:

- You want to edit the new object independently.
- You plan on recombining the new mesh with other elements in the scene.
- You want to keep the old mesh as a backup or alternative.

You should use linked duplicates if the following is true:

- You want to create many copies of a single object and don't want to edit them individually.
- You want only one set of UVs and materials for multiple objects.

When you select a linked duplicate, you can see how many users (copies) of that object are using the same datablock by examining its object data properties in the Properties panel (see Figure 5-18). You can also make the object unique by clicking the number icon next to the datablock's name. Making the object unique creates a new mesh datablock that

Figure 5-16: The plants were all made with very simple meshes (shown with a Subdivision Surface modifier applied).

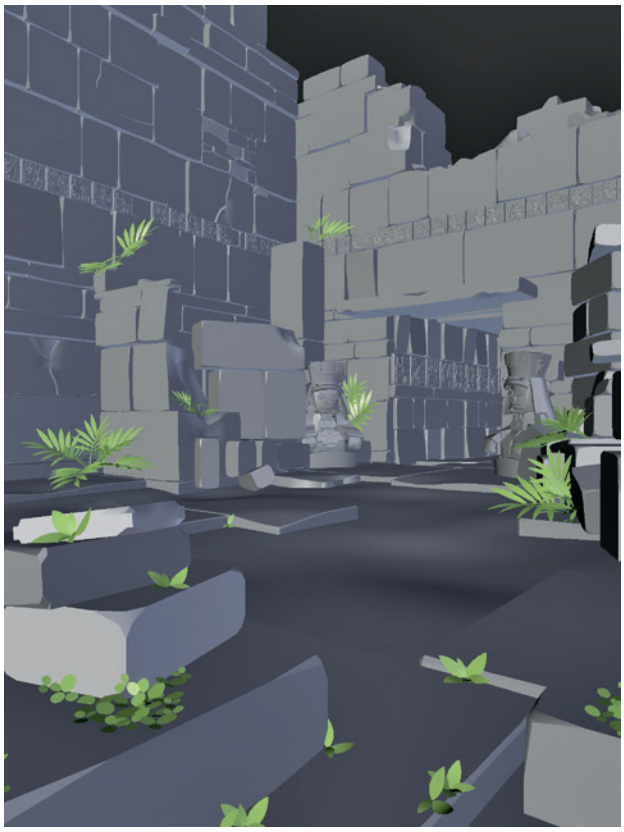


Figure 5-17: Duplicating the different plant components and placing them around the scene. Varying the scale and rotation of the duplicates can go a long way toward making them distinctive.

is now independent of the one it was copied from, allowing you to edit the object's mesh and change its materials separately (as if it were a simple copy).

IvyGen

The IvyGen add-on for Blender is a procedural generator that allows you to quickly create ivy-like vines that creep over your scene (see Figure 5-19). To use it, first enable it from the User Preferences editor (**File ▸ User Preferences**) and then look under the Add-On tab in the Add Curve category. Once you have enabled the add-on, you should have the option to grow ivy on a selected object via the Add menu (SHIFT-A ▸ **Curve ▸ AddIvyToMesh** in Object mode).

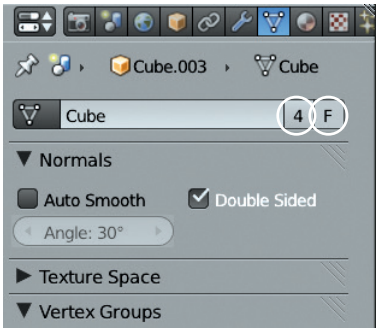


Figure 5-18: Checking the number of users of a mesh datablock. Click the number (4 in this case) next to the datablock's name to create a new copy that you can edit independently. The F icon will create a "fake" user of that datablock, which will save the object and prevent it from being deleted when you save the .blend *file, even if there are no instances of that mesh in the scene.*

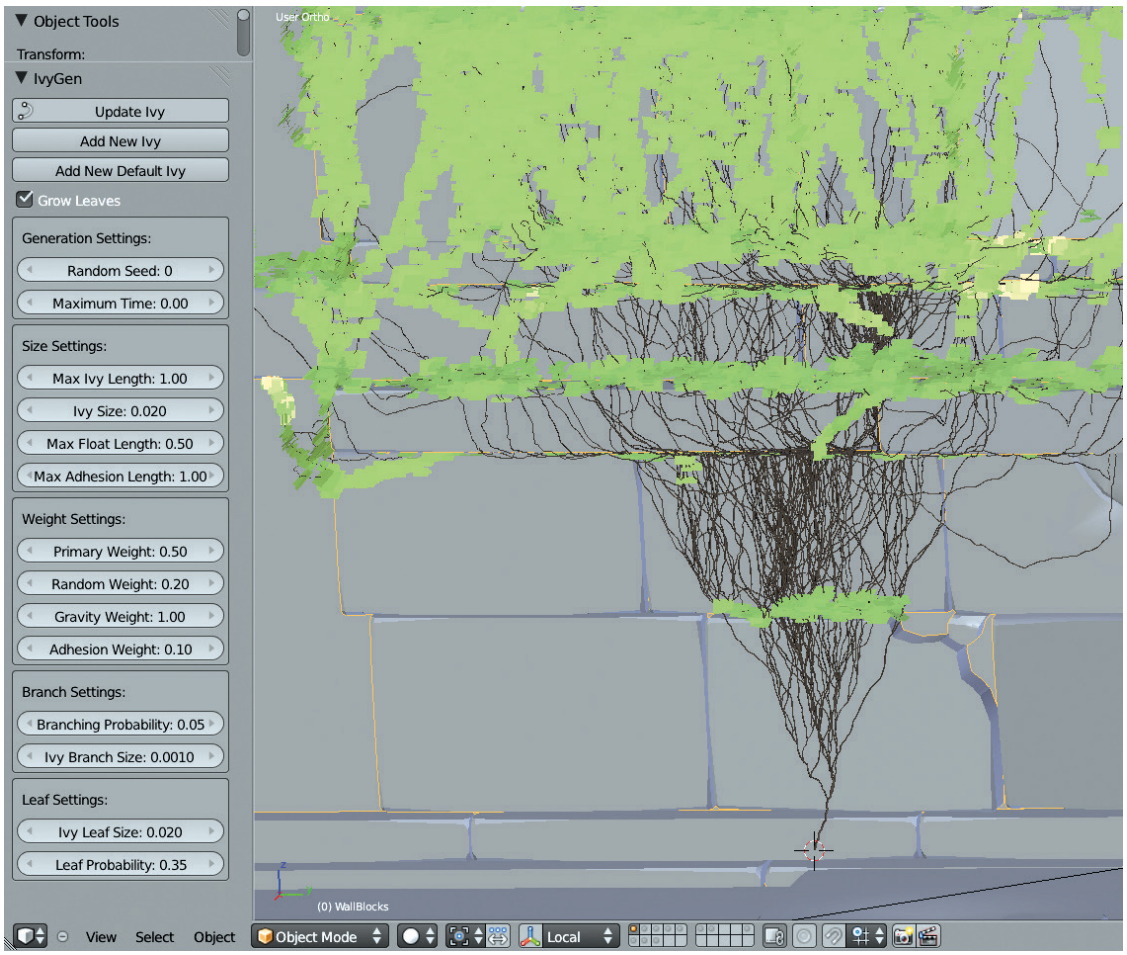


Figure 5-19: Using IvyGen to generate procedural vines. The parameters for how your ivy will grow are in the Tool Options region on the left when using IvyGen. For clarity here, I've added a green material for the leaves and a brown material for the vines.

In order for IvyGen to generate vines, it needs a single mesh object for them to grow over, so we need to create a new mesh that includes all of the geometry we want to grow vines over. To do so, select all of the objects you want the ivy to cover, duplicate them (SHIFT-D), apply any modifiers (using the **Convert to Mesh** operator in Object mode—ALT-C), and merge them into one object (CTRL-J). The result should be one object. If your scene has a high poly count, you might want to skip applying modifiers that increase the poly count a lot in order to give you a lower-poly mesh to grow ivy over; unfortunately, this may come at the cost of some accuracy in how it will grow.

Later, once you've finished growing your ivy, delete this duplicate or move it to another layer so that it doesn't get in your way (**M**).

Next, place the 3D cursor where you want the ivy to start and activate IvyGen with the Add Curve menu (SHIFT-A ▸ **Add Curve** ▸ **Add Ivy To Mesh**). The IvyGen allows you to tweak numerous parameters to determine how the ivy looks, the most important of which are the Max Ivy Length option, which determines how far the ivy spreads, and the Ivy Size and Leaf Size options, which determine the thickness of the vines and the size of the leaves. The leaf probability option determines the leaf density. Other options, like Float Length and Adhesion Length, determine how far the vines can reach out from the wall and how they are affected by gravity.

Keep tweaking IvyGen's settings and pressing **Update Ivy** to regenerate your ivy with any new settings until you're happy with the look of your foliage. Keep in mind that the higher you set the Max Ivy Length, the longer the ivy will take to generate. Also, if you want to cover a large area, it's easier to run IvyGen repeatedly using different starting locations in order to create multiple ivy meshes. For example, in the Jungle Temple scene, I hid the starting locations in the corners of the scene in a couple of different starting locations and let my ivy grow out from there. You can see the final results in Figure 5-21.

IvyGen also creates automatic UV coordinates for the leaves and vines it generates, as

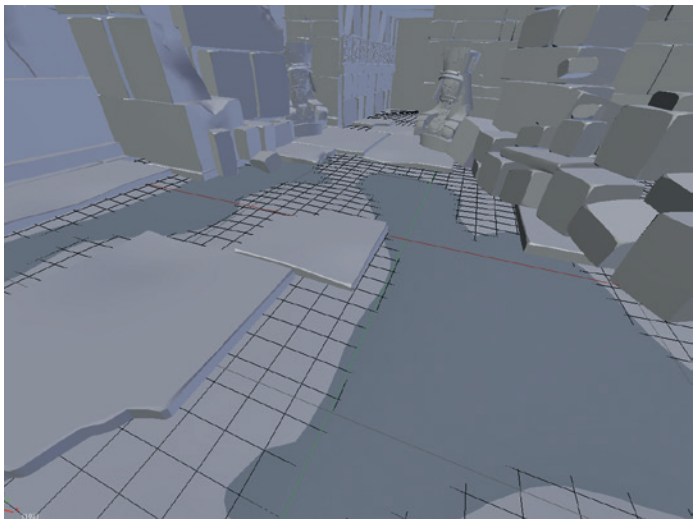


Figure 5-20: Adding puddles to the scene by first creating depressions in the main ground plane mesh and then adding a second flat plane to intersect with it

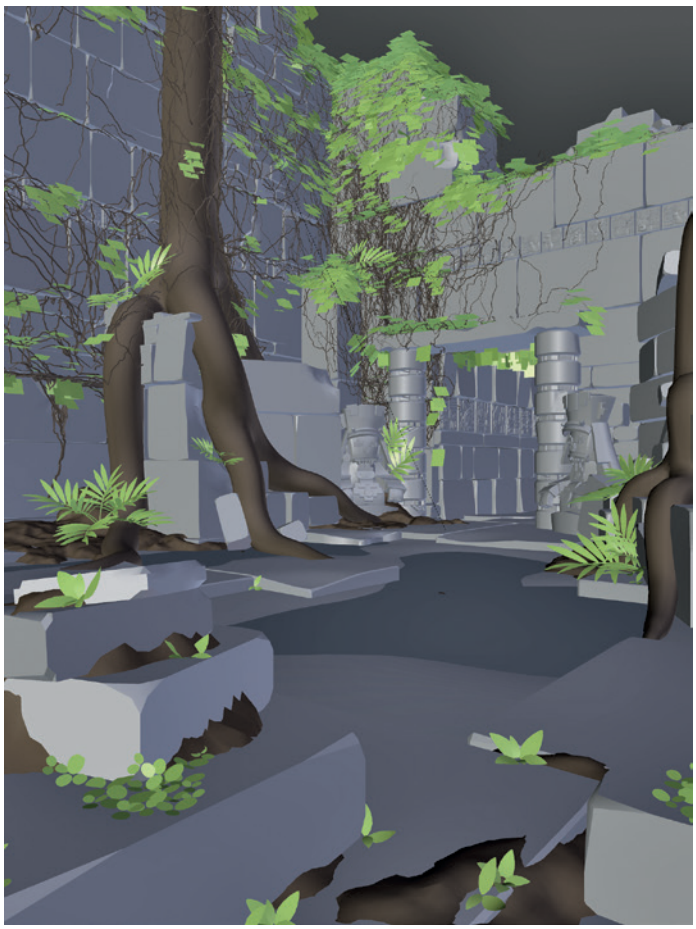


Figure 5-21: The final modeled Jungle Temple scene

well as assigning material slots to them. This feature will greatly speed up texturing and assigning materials to your ivy later. (See Chapters 8 and 12 for more on UV unwrapping and materials.)

Ground/Soil

To make the ground a bit more interesting, I subdivided it a couple of times and roughened it up a bit with the Sculpt tools, which I'll discuss in detail in Chapter 6. Next, I added a new plane (this time keeping it unsubdivided and perfectly flat) and placed it just below the average height of the ground so that some of the deeper areas poked down through it. This produces the effect of puddles on the ground (see Figure 5-20).

Additionally, I created piles of dirt in the corners of the scene simply by creating a plane, subdividing it several times, and using proportional editing to add lumps. By combining this with a bit of sculpting to build up dirt in the cracks and corners between the blocks and other elements, I was able to give the surroundings more of an aged look. The final scene is shown in Figure 5-21.

While the following flows from the modeling techniques already discussed, the parts we are creating fit in alongside the sculpted and retopologized meshes we will be working on in Chapters 6 and 7. You can follow along with this part first or skip forward to Chapters 6 and 7 on sculpting and retopology and then return to this later.

Modeling the Details of the Spider Bot

For the Spider Bot, I needed to create the other mechanical parts of the body that will complete the model when combined with the main body and leg pieces I will be sculpting and retopologizing in Chapters 6 and 7. The aim was to create some feasible mechanical-looking parts, such as joints, wires, and so forth, that complete the look of the Spider Bot.

Joints

The joints were all designed from the same basic template: a cylinder for the central part, which allows them to move freely, with struts coming out that attach to the legs (see Figure 5-22). To produce the struts, I began with a curve object to make the basic shape a 2D curve, used the Extrude setting to give it some thickness, and then converted it to a mesh. Blender's default curve-filling topology is full of skinny triangles, which do not subdivide at all well, so I deleted these faces and filled in the front and back by hand to produce nicer topology (see Figure 5-23).

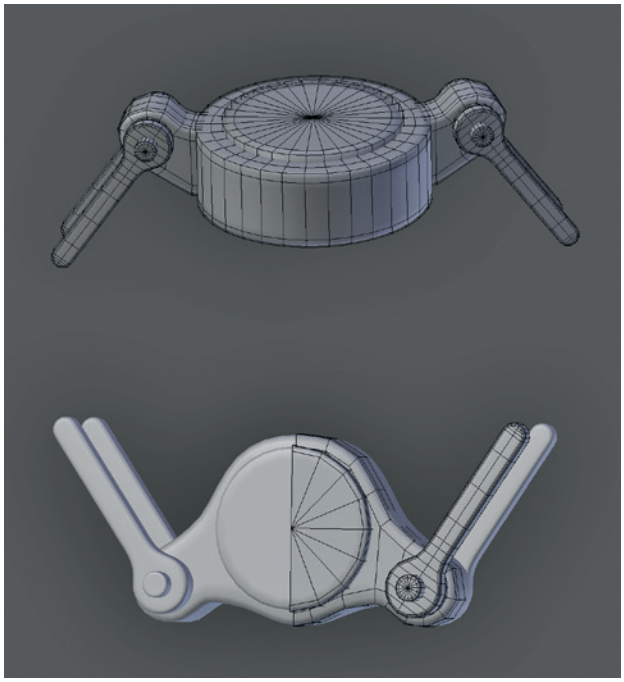


Figure 5-22: The leg joints. Both joints were created using a mix of cylinders for the simple parts and curves (converted into meshes) for the longer pieces. Adding support loops around the edges of the cylindrical parts allows them to subdivide much better.

Wires

The wires are all created from 3D Bézier curves modeled around the legs and other areas to add interest and connect the parts (see Figure 5-24). To add further detail, I converted some wires to meshes to allow me to add some loop cuts and extrusions (see Figure 5-25).

For some of the more elaborate wires, I combined the Array and Curve modifiers to duplicate a single mesh along a curve (see Figure 5-26). First, I modeled a single unit (the ring-shaped object in Figure 5-26), and then I added an Array modifier and a Curve modifier to duplicate that unit and deform it to the shape of the curve.

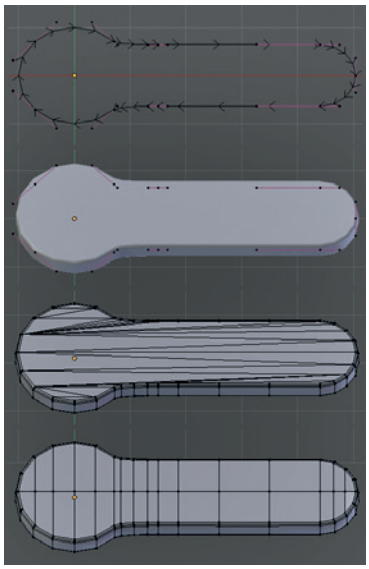


Figure 5-23: Creating the strut elements of the legs with curves and then filling in the flat surfaces of the resulting mesh with cleaner topology

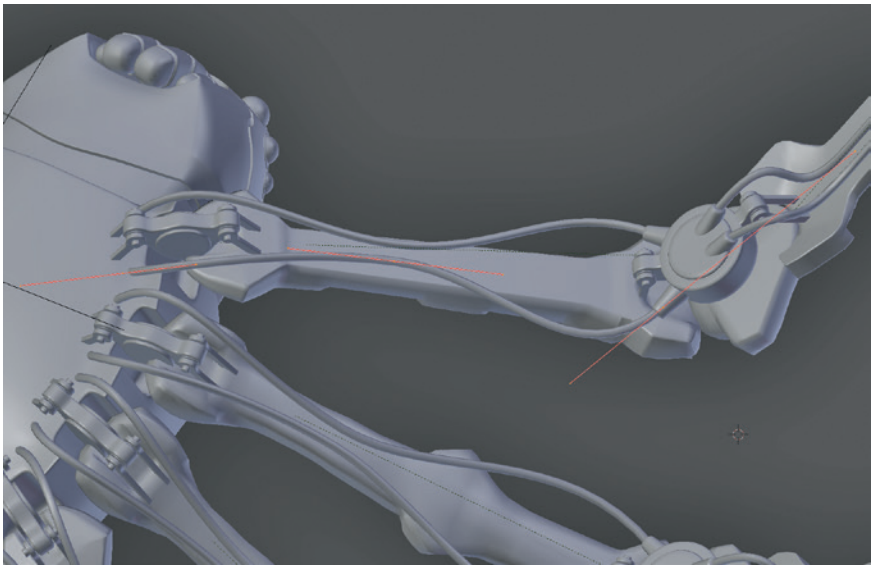


Figure 5-24: Creating the wires for the underside of the legs. These were made with 3D Bézier curves and given thickness using the Bevel setting in the Object Data panel.

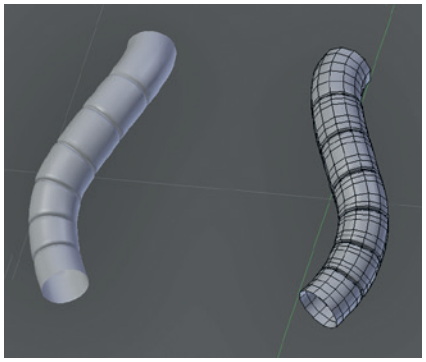
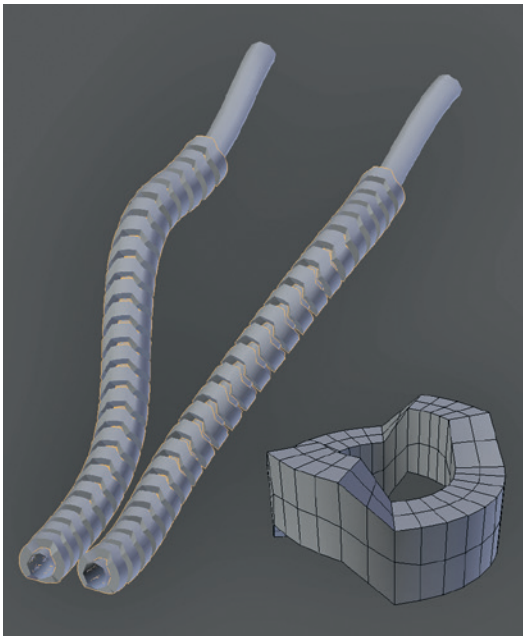


Figure 5-25: Initially, I modeled these tubes using curves. Then I converted the curves to meshes to allow me to add some loop cuts and scale them in to create grooves.



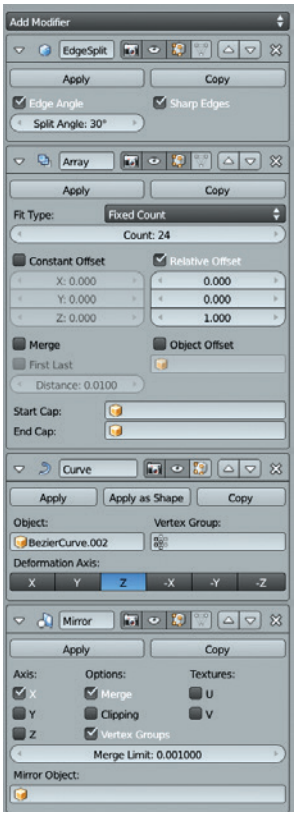


Figure 5-26: A more complex curved object, made by combining an Array modifier to duplicate the base unit (the ring-shaped object) and a Curve modifier to deform the resulting stack along a curve. I also used an Edge Split modifier and a Mirror modifier to mirror the results to the other side of the model.

Coupling

For the coupling between the body and abdomen, I initially created the shape with curves and then duplicated it and converted the duplicate to a mesh (ALT-C). Because Blender's default curve filling creates ugly, long triangles that don't deform well, I fixed the topology by hand by deleting the inside faces and filling in the shape manually (see Figure 5-27).

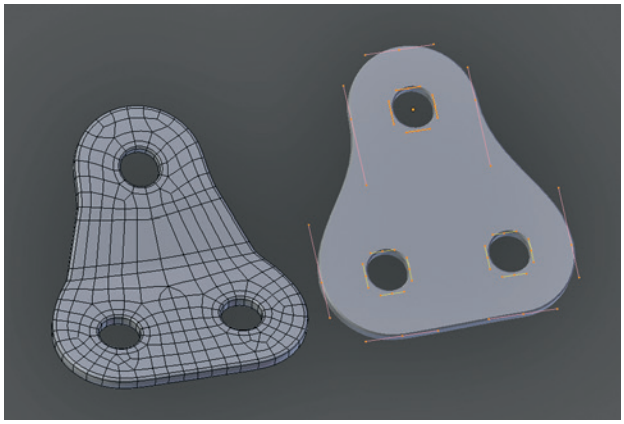


Figure 5-27: Creating the coupling. I converted the curve object (right) into a mesh and then deleted some of the edge loops around the edges to even out the distribution of faces. I filled the inner faces with nicer topology by hand. Then, using proportional editing, I added a bend in the middle.

Other Parts

I placed the Spider Bot's eyes using Blender's Snapping tools: I turned on Snapping to Faces, added spheres in Object mode, and then snapped them to the surface of the head. The fangs are simply cubes, extruded and with loop cuts added to make constrictions where they bend. I added some further embellishments using a mix of Blender's modeling tools and retopology techniques. (See Chapter 7 for these parts and the finished model.)

Modeling the Details of the Bat Creature

The final Bat Creature will consist only of one mesh for the body, which we'll discuss in Chapter 6, but it will need eyes, teeth, and fingernails, too. As these wouldn't be sculpted or retopologized in any way, I aimed straight for the final mesh.

Eyes

There are many ways to model eyes, but in general it helps to model some of the internal structure of the eye first to allow the rendered eye to catch light and reflections realistically. My model for the eye (see Figure 5-28) consisted of an outer layer, which will have a transparent material and which makes up the cornea and the reflective surface of the eye, and an inner layer, which will later be textured with the pupil, iris, and sclera (the white of the eye).

Both the inner and outer layers are made in the same way, beginning with a UV sphere (in Object mode SHIFT-A ▸ Mesh ▸ UVSphere) and then using proportional edit to push in the end of the sphere for the inner part or to push it out a little for the outer part to add a bulge to the cornea. For the inner part, after pushing the surface in with proportional editing, I extruded back the most central faces to create a pit for the pupil. For the cornea, I deleted the end triangular faces of the UV sphere and replaced them with a subdivided plane to avoid artifacts when a Subdivision Surface modifier is added (see the left of Figure 5-27). Using the To Sphere operator (ALT-SHIFT-S) can help you regain the spherical shape of the eye after adjusting its topology.

Teeth and Nails

Both the teeth and nails were derived from cubes (see Figure 5-29). To create the teeth, I began with a cube, scaled it down, and extruded from the bottom. By repeatedly scaling down the bottom of the tooth and then extruding again, I was able to refine the tooth into a point. I then positioned and duplicated the teeth and used a Mirror modifier to fill in the other side of the mouth.

For the nails, I flattened the cube a bit, added a loop cut down the middle, and moved it out a little to give the nail a bit of a curve. I then repeated the same process I had used for the teeth, refining them into a point and then placing them by hand and duplicating as many as I needed.

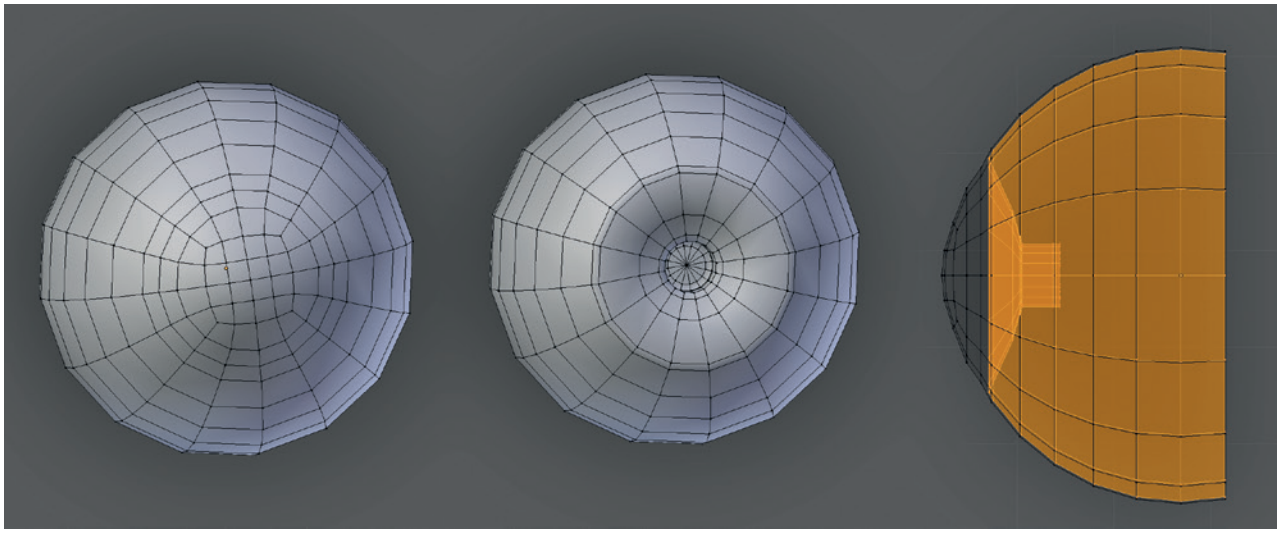


Figure 5-28: Modeling the eye. Left: The outer layer. Note the grid topology at the end of the cornea. Middle: The inner layer. Right: The two combined in wireframe view, shown from the side.

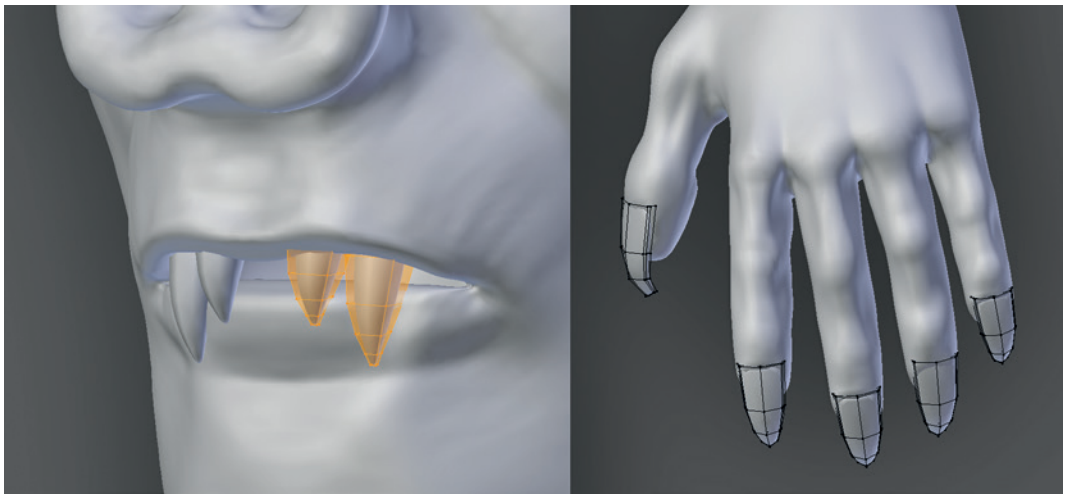


Figure 5-29: Modeling the teeth and nails

In Review

This completes our discussion of modeling the Jungle Temple scene and adding some extra details to the Spider Bot and Bat Creature projects. You've learned how to use a variety of Blender's modeling tools, including modifying existing meshes with modifiers, applying the results of these modifiers so you can edit the results, modeling with curves and adjusting the results, and modeling parts from scratch using primitives and extrusions to build up complex forms.

In the next chapter, we will move on to sculpting in Blender using the Multiresolution modifier and Sculpt tools to create detailed organic and hard-surface forms. In Chapter 7, you'll learn how to retopologize these forms using Blender's modeling tools in order to create your models.