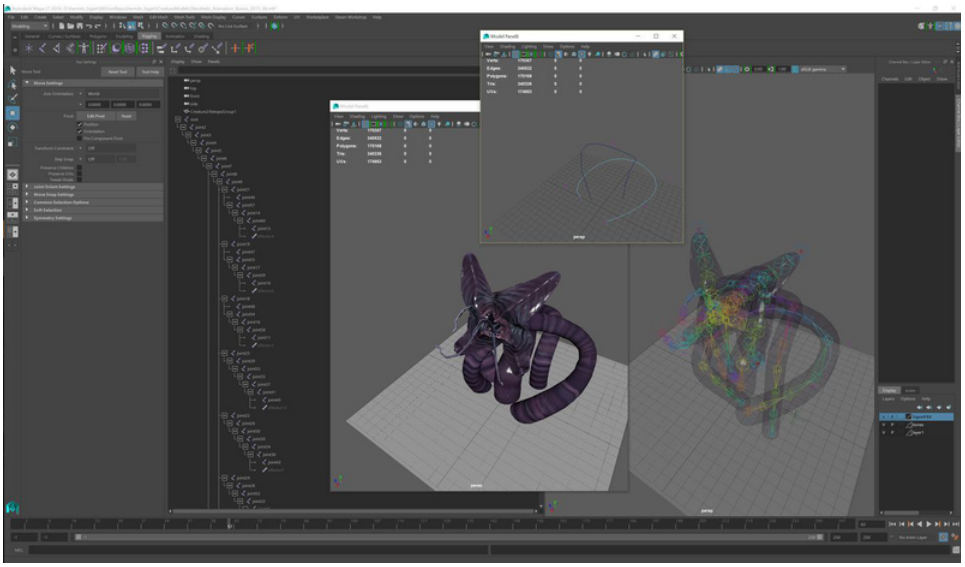
New Creature Process

07/07/2017 NOTE: This process was what was used for this specific creature with the tools we had at the time. It's unlikely we'd continue using Unity for our future projects, but all the same principles apply (rendering the creature on greenscreen with seperated shadows, using an existing creature as a template, etc)

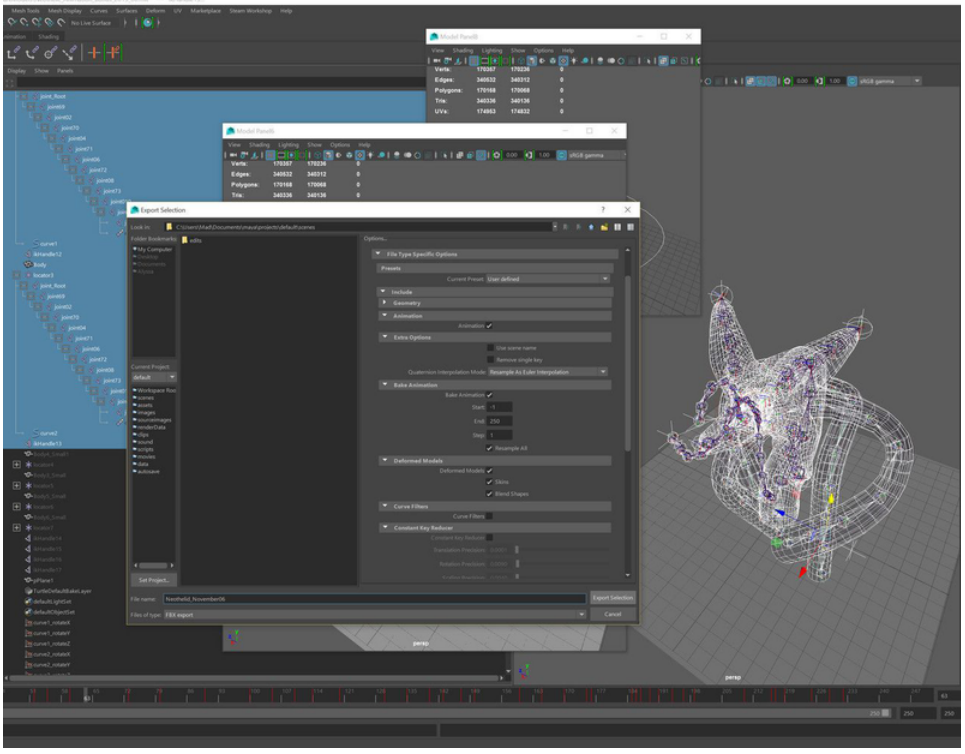
Tools/Code notes



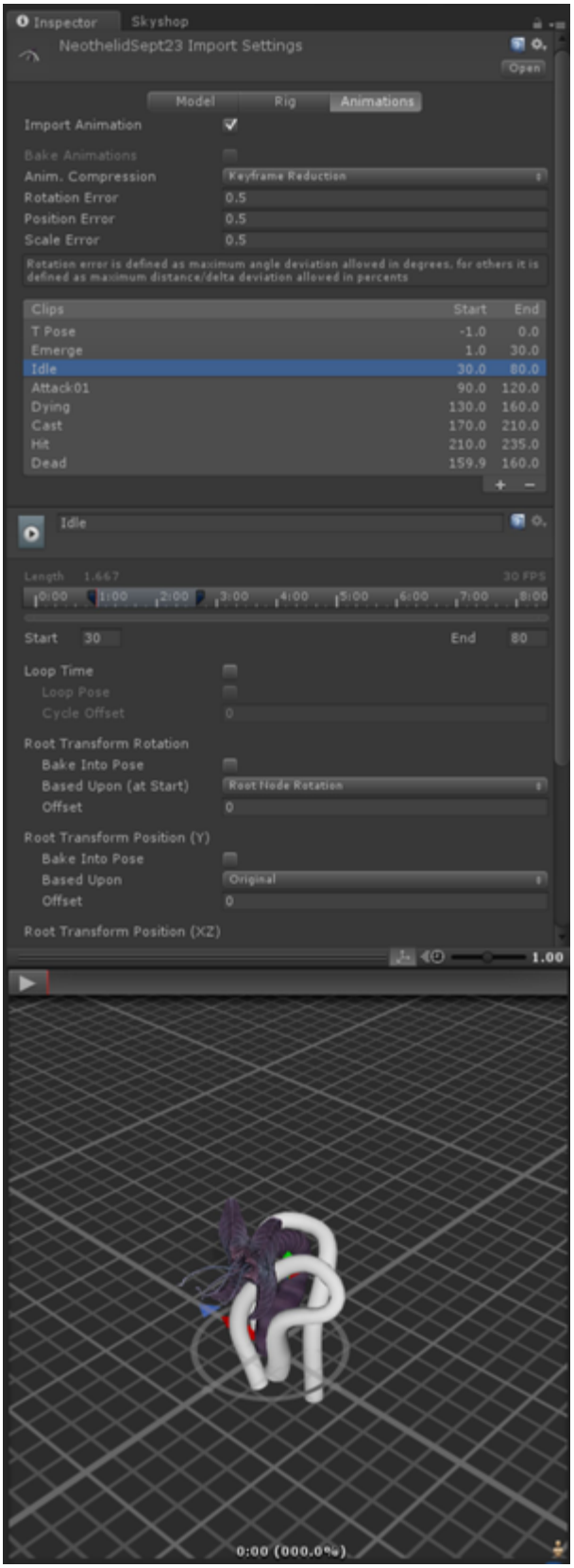
Art Documentation

Maya

The Neothelid's rig is a series of IK chains with spline controllers, and the animation was a combination of bone rotations, IK Handles, and control points.I won't get too heavy into the nitty gritty of character animations, or the specifics of this particular creature, but I will say that Unity has a strong preference for bone based animation and can be weird about constraints or vertex animation. Baking the animation on export to FBX should deal with that, but in the case of vertex animation, they seem to bias towards recommending morph targets instead (maybe Megafiers would be worth it).



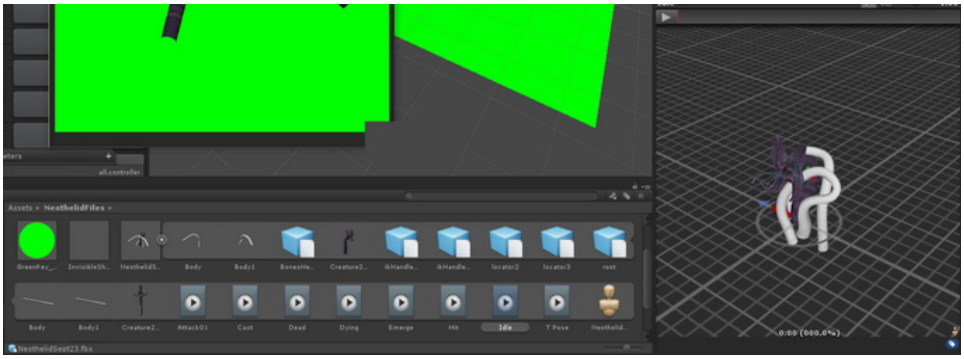
Unity can be fussy about having a keyframe on a bone at times with a large change in position (like the start of this animation, where the root goes from Origin, to deep ‘below the ground’, back to Origin, so occasional weirdness can be fixed by keying bones (especially root) manually. FBX export should bake it down, but doesn’t, so don’t be surprised if there’s occasional breakage in engine.Set up creature animations in one long clip starting with t-pose (keyframing in and out frames at the start and end of each animation), baked frames on FBX export. Select everything relevant (mesh, IK controllers, etc) on export.This might need several tries, but hopefully some of the info of what to expect up ahead will root out some possible problems.

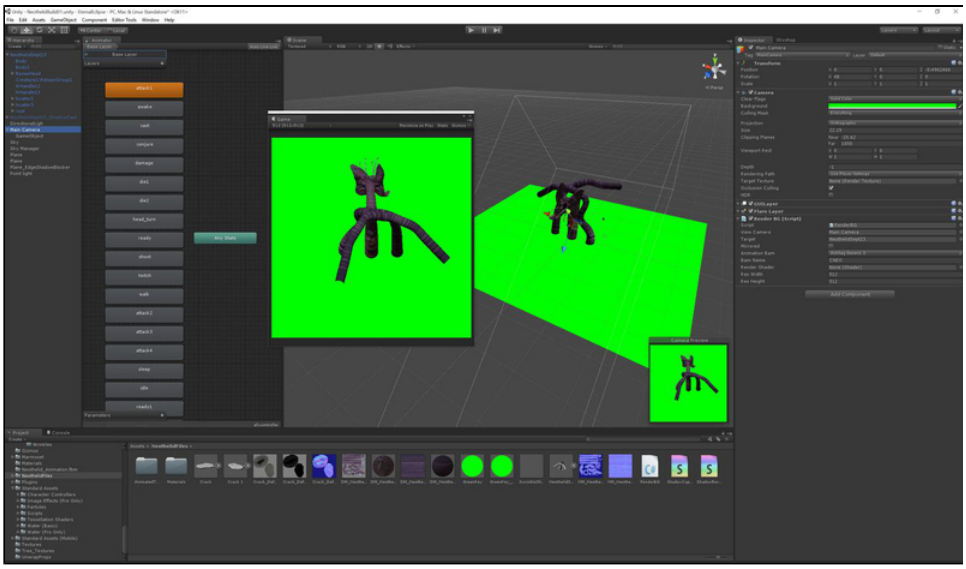


Model Import Settings - Unity Side

Unity has animation in one long 'clip', cut up into sub-clips once imported. Define the start and end points, and name according to whatever existing creature the new creature is based on. In this case, the Neothelid has the same moves as the Ankheg.One part of seeing the mesh in Unity that's a little weird is the difference between the clip preview window and how the engine plays back animationThey sometimes don't match. Sometimes this means the export was wrong, sometimes that's normal.If there's a difference between the clip preview before splitting it into sub-clips and

the preview of the sub-clip when selected by itself, it's likely an issue with bones on export not being keyed properly (the mesh will be shifted in a weird direction, stop in the wrong spot, etc). The best way to avoid this, as mentioned earlier, is to manually key the mesh's bones on the in and out of each pre-determined sub-clip.For example, the Neothelid has an emerge animation where the root shifts dramatically. All bones and controllers are keyed manually at frame -1 (the t - pose used for rigging), frame 0 (the very start of the emerge animation), frame 30 (when the Neothelid settles into the end of the Emerge and start of the Idle, which is the same start frame for each of its animations), and so on. This shouldn't interfere too much with the animation process as a whole, I pretty much locked in those keys as a final step before exporting.Once the clips are defined, the model is ready to be placed in the scene and have it's animations hooked into the script we've created to capture animation frames.

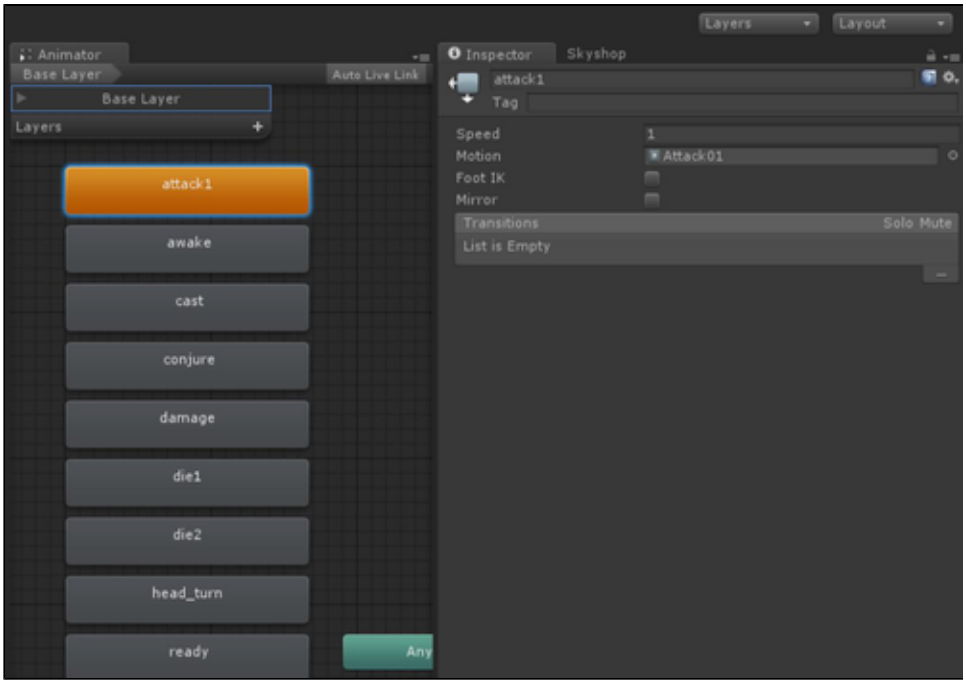




Unity Scene Setup

The Unity scene is essentially a green screen where the model is told to animate, spin, and animate, while the camera captures the images. There are a series of tweaks needed on the Scene side, the Project side, and the scripting side to get it to produce a useable set of images. Yo u'll notice that both the Background (as defined on the camera) and the plane beneath the creature are a keyable colour, and anti-aliasing is off. The plane represents the cutoff of the ground itself (hiding 'subterranean' tentacles and the creature when it's below 'ground level).Make sure that both the

‘ground’ plane and the background are exactly 0, 255, 0, 255. Even a single value off can make a difference. Anti-aliasing is a postprocess which blends pixels, so it must be turned off. When it blends the background in with the creature, we don’t have a sharp edge to cut out from.In this scene, I created a generic skybox for ambient light/reflected light (using Marmoset Skyshop, but this is an external plugin). I also have a directional light for casting shadows, which will be covered in the shadows section.The camera position is partly dependant on the height and size of the creature being captured, and the size of the image being generated. BG style camera dictates a 45 degree angle, and the creature facing away from the camera in the game preview ensures it's roughly facing in the right directions during the frame capture.



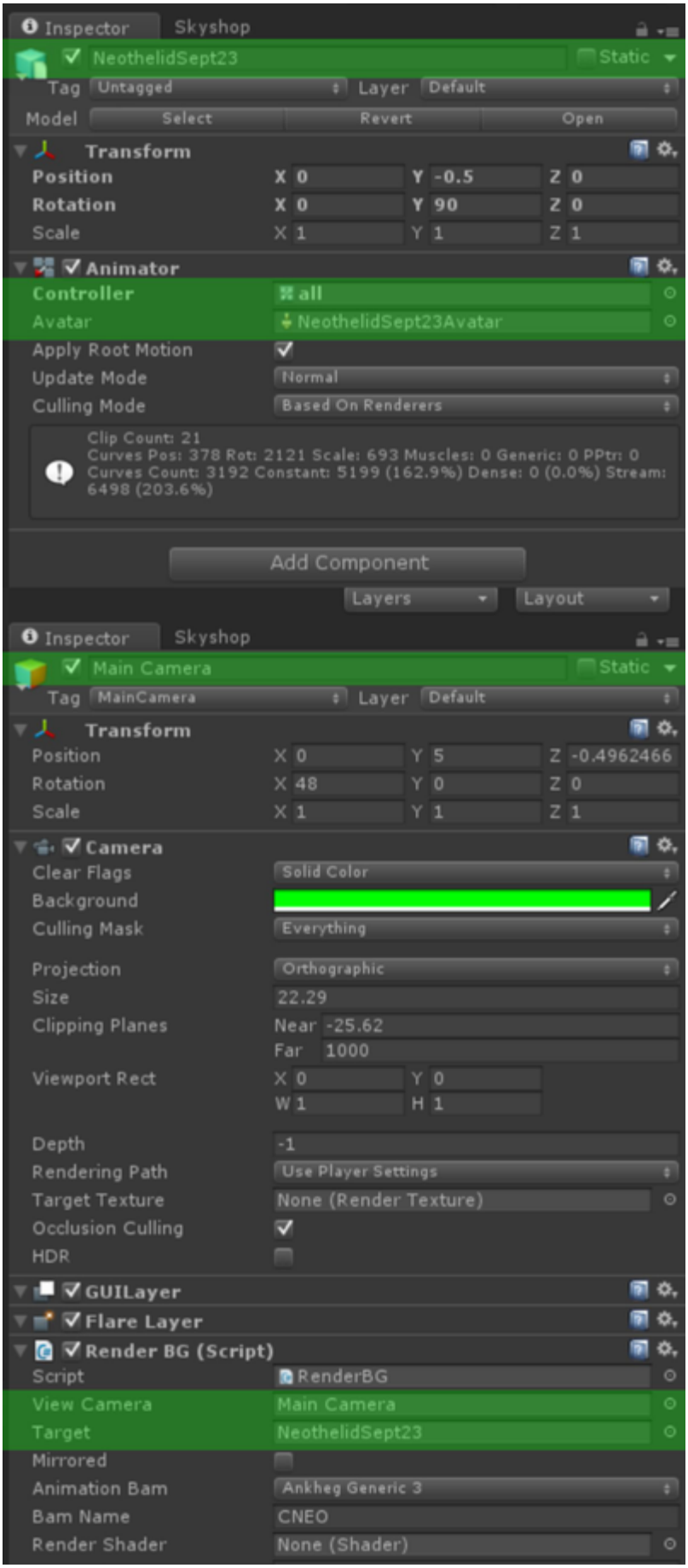
Animator Settings

The animations defined in the Creature's clips need to be plugged into the Animator. You can drag them into the Motion slot. The Neothelid's Descend animation is just the Emerge animation with the speed set to -1, which plays the animation backwards at a speed of 1, the speed all the animations generally should be.RenderBG's definitions are currently based on existing moves and animations that exist within either BG or IWD. The clips can be named however, but once they're inputed into the Motion parameter in the Animator, the assume the name used within the RenderBG script, and are called based on which ones are in it. The animation bam parameter on the Camera being used tells which animation set is used; in this case there are 3 Ankheg BAMs, and selecting this and hitting play would output the frames and BAMs of Ankheg Generic 3.

RenderBG Script

Created by Scott for this purpose, the RenderBG script tells everything in the scene to animate, take screenshots, and output BAMs to the Render Out directory. *07072017 Note: If handled with a 3D package or different engine and not having it scripted, you would generate the BAMs manually from the frames afterwards. Using Unity is really just having access to scripting these actions, and if you were using Maya could likely by MEL or Python.* The only thing that needs to be changed within it on the art side is the definition of how long an animation plays for. Not every Idle is 30 frames,

for instance. These numbers can be changed under AnimationInfo per animation.



07/07/2017: The shadows were captured separately from the model using a greenscreen material that switched from receiving shadows to not, and flipping the creature's visibility (while still allowing it to cast shadows, which was handled in a material applied to the creature as opposed to just hiding it). This allowed us to capture shadows in a separate set of rendered frames from the creature, which is how BG handled it. The shadows needed to be fully opaque, thus the one strong light source.

In the case of this creature, the frame names, material and shadow swaps were done manually for each of the renders before starting the capture. This wasn't ideal (you could easily forget to swap something required for that pass and have to restart the capture repeatedly), but got the job done.


RenderBG.cs
HiResScreenShots.cs
Marmoset Bumped Specular IBL.sh
RenderBG ▸ Start ()
Animations = new AnimationInfo[] {
new AnimationInfo("attack_xbow", 15),
};
break;
case AnimationBam.CharacterSling:
numPositions = 8;
currentPosition = startPosition = 0;
endPosition = 5;
bamdExtension = "SS";
Animations = new AnimationInfo[] {
new AnimationInfo("attack_sling", 15),
};
break;
case AnimationBam.AnkhegGeneric1:
numPositions = 9;
currentPosition = startPosition = 0;
endPosition = 9;
bamdExtension = "G1";
positionScale = 180.0f;
Animations = new AnimationInfo[] {
new AnimationInfo("walk", 1),
new AnimationInfo("die1", 30),
new AnimationInfo("die2", 30),
new AnimationInfo("ready2", 50),
};
break;
case AnimationBam.AnkhegGeneric2:
numPositions = 9;
currentPosition = startPosition = 0;
endPosition = 9;
positionScale = 180.0f;
bamdExtension = "G2";
Animations = new AnimationInfo[] {
new AnimationInfo("ready1", 50),
new AnimationInfo("emerge", 30),
new AnimationInfo("hide", 30),
};
break;
case AnimationBam.AnkhegGeneric3:
numPositions = 9;
currentPosition = startPosition = 0;
endPosition = 9;
positionScale = 180.0f;
bamdExtension = "G3";
Animations = new AnimationInfo[] {
new AnimationInfo("attack1", 30),
new AnimationInfo("attack2", 40),
};
break;
case AnimationBam.IWDAttack1:
numPositions = 8;
currentPosition = startPosition = 0;
endPosition = 5;