AS12634

VR: Designing the Experience

Scott DeWoody
Gensler

Learning Objectives

- Discover experiences and the storytelling of VR
- Review VR Solutions and Applications
- Review best practices when designing the experience
- Learn how VR helps with design and the bottom line

Description

In this course, we'll cover the rules and techniques for generating experiences for virtual reality (VR). We'll go through the lessons learned in VR at Gensler, along with the different kinds of VR that are available to designers today. From 360º rendering to room-scale VR, there are plenty of options to dive into.

Speaker(s)

Scott DeWoody has always had an affinity for art and technology. After seeing the animation being done through computers, he knew he could combine the two. In 2007 he graduated with a bachelor's degree in media arts and animation. He focused on lighting and rendering techniques using 3ds Max software, V-Ray for 3ds Max, and Adobe Photoshop software. A day does not go by when he is not using one of these applications. Image quality and workflow are the top priorities in his work. He is constantly studying color theory, composition, and new ways to produce the most effective possible results. He has worked at M. Arthur Gensler Jr. & Associates Gensler, for the past 8 years as a visualization artist and manager. He has worked for numerous clients, including NVIDIA Corporation, ExxonMobil, and so many more. Currently, he is exploring the new possibilities of architecture in the interactive space with gaming platforms, augmented reality, and virtual reality.

AUTODESK UNIVERSITY

The Honeymoon Phase

As of March 29th2013, the industry has been captivated by Virtual Reality with the release of the Oculus DK1. Fast forward to the day of this presentation, November 14th 2017, and it's been a little over 4 years of Virtual Reality in our industry. In that time, we have seen the full release of the Oculus, HTC released their own HMD, PlayStation launch their own HMD, and Microsoft even start to get into the VR Space with Windows and the OEMs making their own HMDs. VR is even slated to hit $7.2 Billion this year alone. But the one thing we haven't seen yet is that one piece of VR content that makes us WANT to buy an HMD. Sure, we all realized the potential for VR the first time we tried it, and we also realized that this is probably the future of our industry. But if we take a step back, it's difficult to name any one experience that taken the world by storm. What VR needs is that "Pokémon GO" experience. Something that just captivates everyone and brings the technology mainstream.

"You have to start with the customer experience and work backwards to the technology." – Steve Jobs

For the past four years, we have been extremely focused on the technology and demos to show off the potential of the technology. But no one has taken it past this stage yet. With all my traveling around the world, the presentations I've given on this subject, and the presentations I have witnessed on this subject, we are relying too heavy on the technology and not the content. Everything I have seen has just been "Look at me change this color" or "Look at me point a laser here so we can all talk about it"… And the list goes on. Now these aren't bad things, is just the industry seems to be stuck here.

'Every technology that comes into filmmaking is first a gimmick – it takes a while for filmmakers to understand how to use it.' – John Lasseter

And it's not that people haven't tried to progress. There are a lot of experimentations out there. But those who are trying try seem to be forcing a square peg into a round hole. (And if you're one of the few working on something against what I am talking about, I want to see it!) I believe by this point the industry is starting to realize that you cannot take traditional methods, and cram them into VR. But what is next? What happens when traditional methods of storytelling do not translate? And this is the key. And I'll let everyone on a secret:

No one knows the answer to this. Including myself. But we can all help each other get there!

I have some good ideas around what we need to be focusing on, which is why we are all here reading this document. But it is this next step we all must take, which will launch VR into its next phase. Because the Honeymoon Phase for VR is over. It may not seem like it, but it is very much finishing up this next year. The technology is out, and the advancements have slowed down more than they have in the past 4 years. VR Techniques are well documented online, and there is plethora of VR at every conference now. Just look at the AU 2017 class schedule. You can't scroll through it without seeing something that says VR. All of this is great, but a majority focuses on the technology and workflows… Which are important, we've just hit the full saturation in the industry. And I'm going to assume you're reading this partly due to that. That we all know, there is something missing. That VR cannot just be HMDs, changing colors on objects, and walking around. So, let us talk about this a bit more in-depth!

The Three Types of Experience

There is one thing that no one is talking about, and that is the **Experience** in VR. Experience is the real reason why we all went “Whoa” (Imagine Keanu Reeves) the first time we put on an HMD. And that’s all because we realized the experience that VR could give us. We were just introduced to a technology that could transport us to a world that literally does not exist, and we can experience that world as we would in reality. It wasn’t about what brand the device was, or what GPU it is running on, or what game engine is the best to work with. It was about that “Whoa” moment. But again, that first reaction only lasts so long. Eventually we will want more, especially our clients that we are designing all this content for.

There are three key areas in Experience we all need to focus on to create a successful VR Experience.

Storytelling

This will be the portion that brings everything together. So, it is very important to have something here. Even if the story is small, the slightest reason to give someone an understand of why they are in your world will help ground them in the experience.

User Experience

This is probably the most talked about “Experience” in the industry, because it coordinates with making sure people don’t get Motion Sick during Virtual Reality. There are even “Comfort” ratings for VR Games now because of this. Though User Experience should not be limited to this. How someone interacts in your VR World is extremely important. Does it feel natural? Is it intuitive? Can my Grandmother do it without any help? All important questions to be asking during development.

Visual Experience

The Art Direction of your VR World will help greatly ground your story, and your user, inside of the experience. Although our first instance is to make everything look as real as possible, that does not need to be the direction. Plenty of stories and experiences are told in a stylized manner, an VR is no different.

Do not think of these three points as being equal in your VR Project. Some projects will probably use a little less story, and focus more on the UX/UI. But all VR Experiences do need all three of these to make something complete. Because when all three are considered, you have something that does not look like a Tech Demo any more. And that is what we all must move away from making.

‘Every movie has three things you have to do – you have to have a compelling story that keeps people on the edge of their seats; you have to populate that story with memorable and appealing characters; and you have to put that story and those characters in a believable world. Those three things are so vitally important.’ – John Lasseter

Storytelling

People have been telling stories longer than any history record can show. Storytellers were once very well respected people as they could help share information about faraway lands and experiences people would never dream of going to. The earliest recording of storytelling can be traced back to a tale of Gilgamesh in Egypt.

One thing is clear; storytelling is a part of being Human. It's in our DNA. Just look at all the forums of medium we now have for telling stories: Movies, Theater, Books, Radio, Podcasts, etc. There are whole industries around these formats, that have multiple sub-industries tied to them. Virtual Reality is no different, it is just a new form of medium to tell a story.

Your VR Project will greatly determine the scale of your story, but you should always have one. So, the first part of creating a story is probably the most important:

Concept

This is part will be the backbone of the entire VR Experience that you are trying to design. **DO NOT SKIP THIS STEP.** Without this, your experience has zero purpose and it will show. The biggest question you should probably start with is:

> **What am I trying to communicate to my audience?**

Write your idea down, and read it out loud to yourself. Does it make sense? I know that sounds a little strange, but it works. Also, don't be afraid to run the idea past someone who has zero involvement, or who might be in your target audience. If it makes sense to them, you're probably on the right track. You can follow this question up with:

> **What are the story elements?**

This is it. This is what everything is about. This is the reason people want to try your VR Experience. So, spend some good time thinking of exactly what you will be telling your audience. Will it be an epic story? Will you make it a mystery? Is the story left up to the user to think up the missing parts? Is there narration? Are there any characters? Then ask yourself:

> **Is there another way to communicate this idea?**

Do not be afraid to think of a few alternative ways to tell your story. The first idea you have may not be the best. So, iterate and brainstorm! Spending time here will help so much down the pipeline. The last thing you want is to have a Eureka moment during development that could derail everything. And then comes the big question:

> **Is VR the right medium for this?**

I know this might sound a little counter to what we're talking about in this document, but the question needs to be asked. Do not do VR just because you think you should, or that you need to do it because everyone else is. Do Virtual Reality because it makes sense to what you're trying to communicate.

Choose the right purpose, people will be attracted, motivated, and unified. – Robert Wong, Vice Present at Google Creative Lab

Once you get through the Concept Phase, it is time to move on to…

Storyboarding

"A storyboard is a graphic organizer in the form of illustrations or images displayed in sequence for the purpose of pre-visualizing a motion picture, animation, motion graphic or interactive media sequence." – Wikipedia

This process is going to take all your concept ideas so far, and it will force them all down onto paper. Storyboards to not need to be works of art, but they do need to communicate the flow of your experience. This step is critical, and should not be skipped either. There is no wrong way to do a storyboard. Just make sure everyone in the development pipeline can understand it. Pre-Visualization plays such a major party in any development, and VR is no different. You can even take it a step further, much like the movie industry does, and roughly block everything out visually in 3D. But this needs to stay incredibly rough, and no –real- development should be done. Only enough to get the ideas out of your head and into a format that is easy to understand visually.

Project Roadmap

Once you have completed the Concept and Storyboarding Phases, this will give you an excellent view of what you will need to create to complete your VR Experience. You should be able to derive the following:

- What do I need to model?
- What audio do I need to record?
- Do I need a Script?
- Do I need more money to complete this?
- Do have enough time to complete this?
- Do I have enough people staffed?

The "checklist" that can be generated now will help keep production on track, and should educate all the key players who will be involved with the production. It's extremely important to get everyone up to speed with this, and have all key investors of the project to sign off on all this prior to production starting. This will help mitigate anything that goes off track after this point. Especially when those changes could cost money and time.

'The way the films look will never entertain an audience alone. It has to be in the service of a good story with great characters.' – John Lasseter

User Experience

When creating content, be empathetic above all else. Try to live the lives of your audience. – Rand Fishkin, Founder at Moz

With VR being an interactive medium, the User Experience becomes a very large component in its design. And unless you have a heavy background in UX/UI Design, this will probably be the biggest challenge in your VR Experience. It is going to be very easy for us to get lost in this area of design. All of us know the technology, and we are also extremely invested in our designs. It is best to do a lot of testing around your UX/UI for the experience. This will keep yourself focused on how it is coming together versus how you think it should come together.

A few key rules to live by for UX/UI in VR are:

Keep It Stupid Simple (K.I.S.S.)

A rule that honesty applies to anything in life, but especially to user experience. Too much of anything is a bad thing. So, if you can consolidate the experience, condense menus, and have less "click", the better off you will be. Not only will it be simpler in the end for the User, but it will also be less work for you implement. Not to say that less isn't always easier though.

This is some of the main ideology that has gone into designing the experience we all have on our mobile devices. How irritated do we all get when something takes too long to find, or we must dig through menu after menu? The same happens in VR.

Rule of thumb for UX: More options more problems. – Scott Belsky, Vice President of Products & Community at Adobe

UI needs to be self-evident

Running right off the K.I.S.S. method, the User Experience needs to be obvious to the user. This will be heavily dependent on the target audience you are aiming at. If you're targeting a wide audience, who may consist of people who have no experience with VR, it will be important to make your UI more up front. If you target more advanced users, you can probably get creative with the UI. But no one should ever wonder where something is, or how to interact with the UX/UI.

It will be very important to test this thoroughly with someone from your target audience. The last thing you want to be doing in a client presentation is focusing on how to use the VR experience with someone who has no idea what they are doing. It will completely de-rail the experience you are trying to give.

Also, don't be afraid to add in help menus if needed. Just make sure that they aren't obtrusive to the over-all experience. You'll want to stream-line them in to make them apart the overall experience.

Text needs to be legible

"Objects in mirror are closer than they appear." – Your Car's Side Mirror

Scale in VR is not what it will seem in the editor that you are working in. Make sure that you thoroughly test your UI in the target platform that you are delivering your experience. Every device has its own resolution and format, and you will need to design directly to that platform. Do not wait to do this at the end of production. Blocking in the Text for your UI should be one of the first things you set up.

Interactions need to be clearly visible

When adding interactions to the VR Experience, you will want to make these extremely clear to the user. Any confusion here, and it will start to pull the user out of the experience. This can be done in a variety of ways, and should be linked heavily to your Visual Experience.

Thankfully video games have been doing something like this for years, so there is plenty of inspiration to pick from. A good example is how Naughty Dog treated hidden objects with a slight sparkle that would show up every few seconds on screen. Something subtle enough to get the player's attention.

So this doesn't have to be anything extremely complicated. A simple highlight, or icon, around the object that can be interacted with could be enough. This will vary from experience to experience.

User Interface should feel natural with the input controllers

This will be heavily dependent on the input device you will be using for your VR Experience. Most of the major companies who design these devices put a lot of thought and effort into the design for the user. This will help a lot in designing the UX around these devices, but it something that needs to be closely paid attention to. While in VR, if anything feels strenuous, or un-natural, it will break the experience easily. Most input devices have a wide range of inputs that can be used, so the K.I.S.S. principal applies greatly here.

There has already been a lot of work done in the industry around this, so inspiration should be easy to find. A few key examples are:

a. Triggers are good for picking up and holding objects
b. Touch pads are good for swiping through menus
c. Triggers are also good for clicking through menus
d. Holding two buttons at the same time, but one in each hand, has proven good for object/scene manipulation
e. Controllers can also be considered "Solid" Objects in the game world, and effect other objects. Think pushing something around without having to click, or pushing a button to activate something.

Visual Experience

It is extremely important to establish the visual look and feel of the VR Experience before diving into the main development of the project. This will greatly impact every part of the experience and development process, so having it locked down before starting is key.

Most of us in Design will probably lean towards the “Photo-realism” end of the visual spectrum, as we’re looking to show our designs in the most realistic way possible. But design can still be communicated effectively in a more stylized way. This is especially true in earlier phases of design. A good rule of thumb to remember is:

The more realistic you go, the more work you and the computer is going to have to do to achieve the high visual quality. While something a little more stylized could be easier on the GPU.

Adding in Post Effects such as Bloom, Screen Space Ambient Occlusion, Screen Space Reflections, HDR, etc add to the rendering time for the GPU. And it is SUPER important to keep the rendering time to about 13ms, or 90 frames per second. The slower the rendering, the more likely someone will feel sick from being in your VR Experience. This is due to our eyes perceiving things not as fluid as we know them to really be. This disconnect is what screws with our brain, thus creating the feeling of motion sickness. (Some of these Post Effects are not supported in VR!)

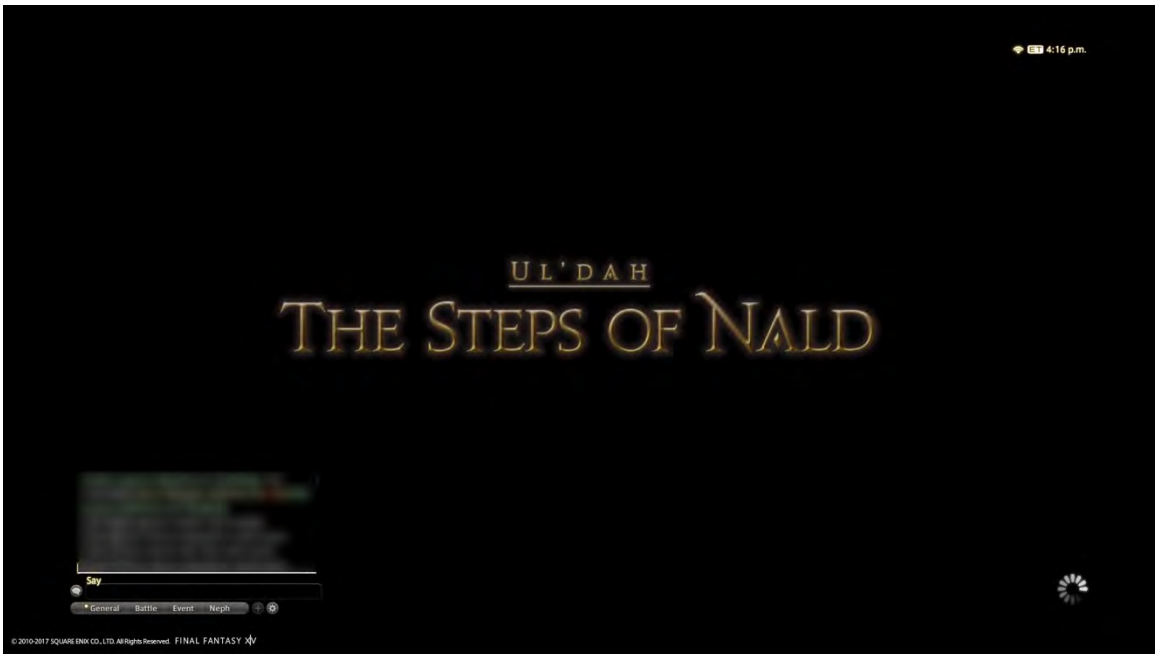
But Post Effects are not the only thing to worry about: Model Quality, Poly Count, Texture Sizes, and Shaders all effect the performance of our VR Experience. And the more complex these get, the more time they will take to render. We will talk more about this a little later in this document. But the Visual Experience will greatly influence the development of all these assets.

The Visual Experience is not just limited to the “Style” which you are trying to achieve in VR, but also how the user will be interacting with the experience. This portion goes hand in hand with a lot of the UX/UI Design, which should be greatly taken into consideration when developing the Visual Experience. But it also doesn’t stop there either. The Visual Experience can create a visual language to help guide users through the experience. These techniques can be seen in most video games today. Some examples are extremely in the user’s face, and some can be extremely subtle that blend into the experience. (The following screenshots were taken during my own gamplay. All content is copyrighted by their creators.)

In your face example: Final Fantasy 14 (Square Enix)

MMO’s are a big deal when it comes to their UX/UI and Visual Experience. In Final Fantasy 14 there are visual elements that blend in with the over-all aesthetic of the game. But one element stands out: The Zone Change Dots. There are floating blue dots that signify the end of an area (known as a Zone), that when cross will move the player to a new location entirely. And while floating blue dots might seem weird, but they are consistently used throughout the game. And then they just become natural for players to see and interact with.

In Final Fantasy 14, those floating blue dots signify an area change. If the player walks through them…

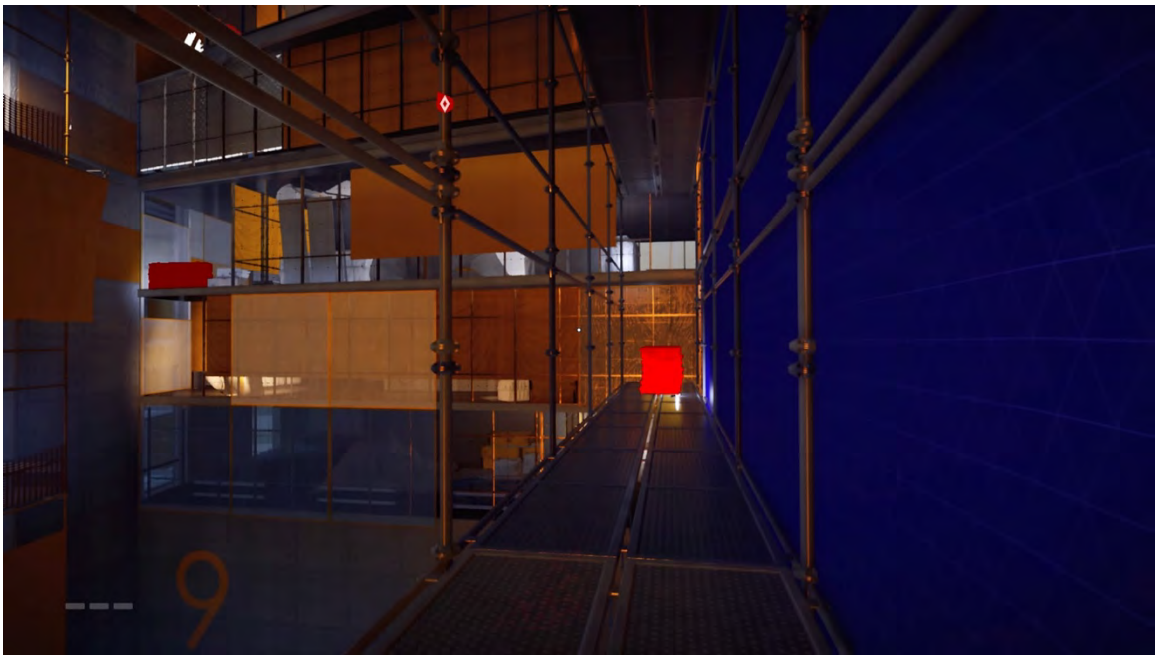


A loading screen appears…

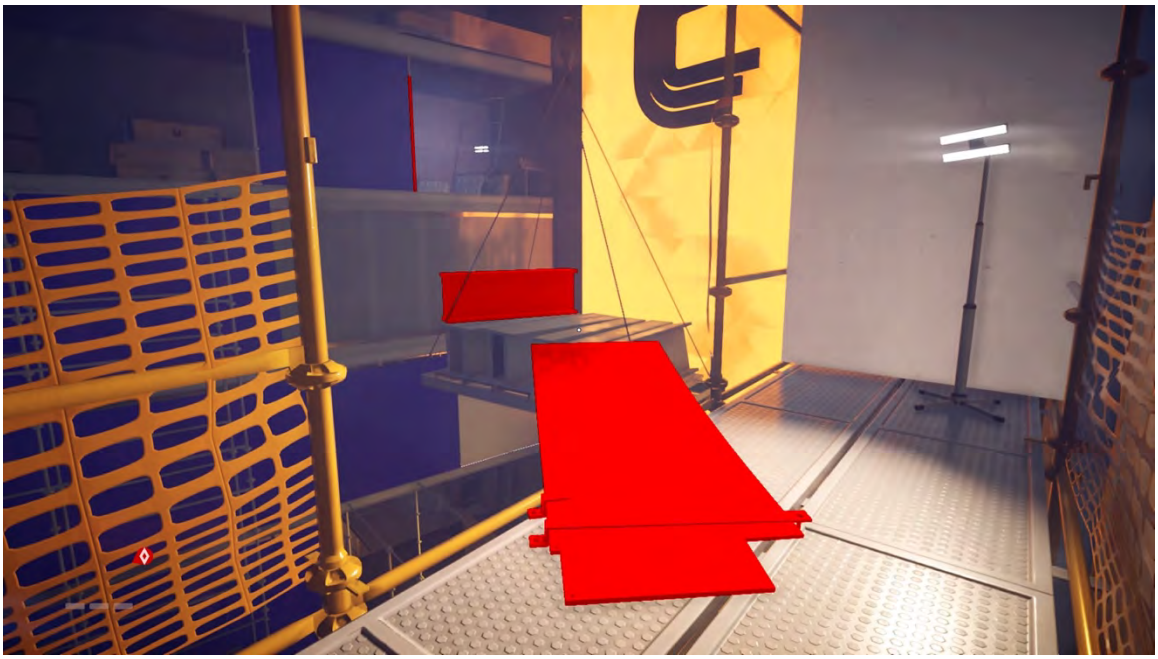
And then the player enters another area!

Subtle, but still in your face some: Mirror's Edge (EA)

This game balances the Photo-Realism and Stylized look and feel extremely well. This is known as Hyper Realism. But one thing the designers keyed in on with this game is that as players are chased through a level, they are going to need to make extremely fast decisions. And there is no time for the player to stop and think about what to do next. So, they designed the key elements to interact with as bright red objects. They call attention to themselves just with their color alone. There's no additional glowing, or sparkles, or anything else of that nature. But there will be a red box to quickly jump over, or a red pipe to climb up or down.



Here is a red glowing stack of lumber the player can climb up, which leads to…



More red glowing lumber, and beams, etc…. but the path is clear to the user where to run next!

Subtle, but you'll see it if you look for it: Uncharted Series (Naughty Dog)

The Uncharted Series is probably one of the most realistic, and cinematic game series, out there to date. Naughty Dog has done an amazing job through and through with this series. They aimed for a completely realistic looking, and feeling, experience. The UI in this game is very minimal, and the level of detail added to user interactions in sometimes just unbelievable. But as you take Nathan Drake through his adventure, he must do some daring acrobatic climbing and stunts. The cues for moving along these paths in the game are blended extremely well into the environment itself. There might be some scuff marks on the wall, indicating you should try to climb here. Or ledges will have some erosion on them that indicate you can grab on to it. Even Drake himself will sort of change his position a little to indicate a change in the path. After playing the game for just a little bit, players become extremely accustom to this visual experience. Which brings the level of realism that Naughty Dog is trying to achieve.

Check out the wood, and window ledge, in this screenshot. Notice the white markings, this indicates to the user that they can climb here.

The same graphic element is then repeated, so the user knows after a while of seeing this that the area can be interacted with.

The take away from this is: Use the Visual Experience to help immerse your users even more into the experience. It is so much more than just how things look, and will make your Experience even that much better. Even if no one can put their finger on why.

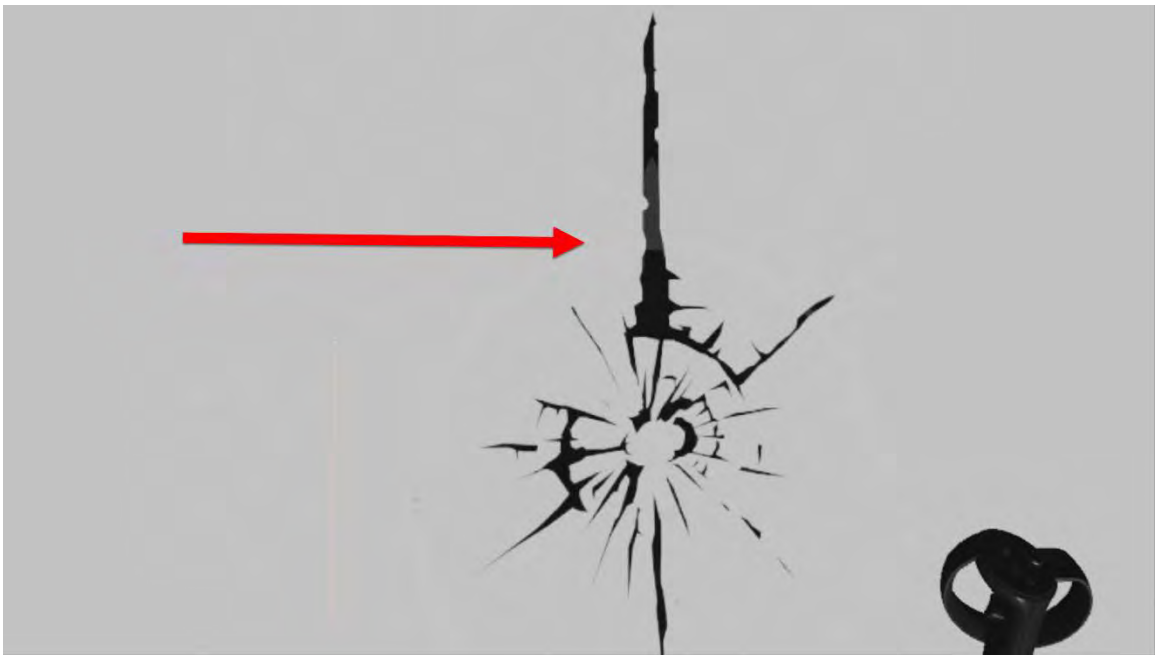
VR Experiences Breakdowns

With about 4 years under the industry's belt now, there are a lot of good, and bad, VR Experiences out there to see. So, in this section I thought we would put the three Experience Types we just covered to the test. We will look at the few VR Experiences that I have personally found to be extremely good, and entertaining. Ones which made me go "Whoa" (Imagine Keanu Reeves Again), ones which kept me coming back for more, and a few that made me feel something. And on top of it all, we will cover the GenslerVR Experience, and our design logic around some of the choices we made when creating our own platform. The following experiences are in no specific order. **(All screenshots are copyrighted by their creators, which are listed below in each section. I obtained these by capturing them during my own gameplay.)**

Robo Recall – Epic Games and Oculus Studios

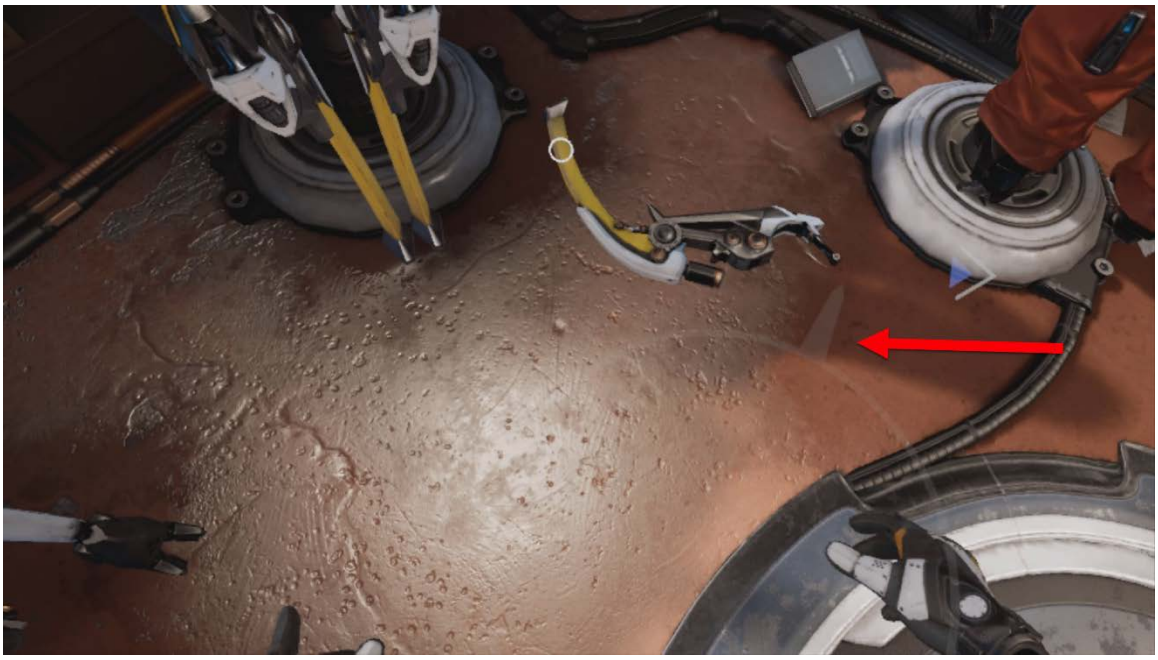
Story: Robots have gone crazy and it's your job to "recall" (aka destroy) them before they terrorize the city!
Visual Experience: Hyper Realistic
Key Design Elements:

1. **Directional Marker On the Ground**
   This is one thing that stood out to me right away. After trying so many VR Experiences, one thing I found always confusing is which way you are actually facing in reality. Robo Recall solved this elegantly by putting a graphic on the ground, which points towards the computer screen! This is important for the game, as the player will be spinning around a lot trying to fight off hordes of evil robots. Eventually players will run into a warning saying to turn around and face the Oculus Sensors. Or even more importantly, when teleporting in this game, the player must pick the direction to face. This direction is always going to re-adjust the player to face "forward", which is where the computer screen is located.



This is the main menu screen's wayfinding graphic that points towards the screen.

This is the Teleport Action inside of the game. Notice the Arrows at the end? This is will determine what direction the player is facing when they teleport.



Here the Wayfinding Graphic again, but this time inside the game. Notice it's much simpler in design, and less obtrusive than the main menu. It's enough to be hidden and not pull anyone out of the experience.

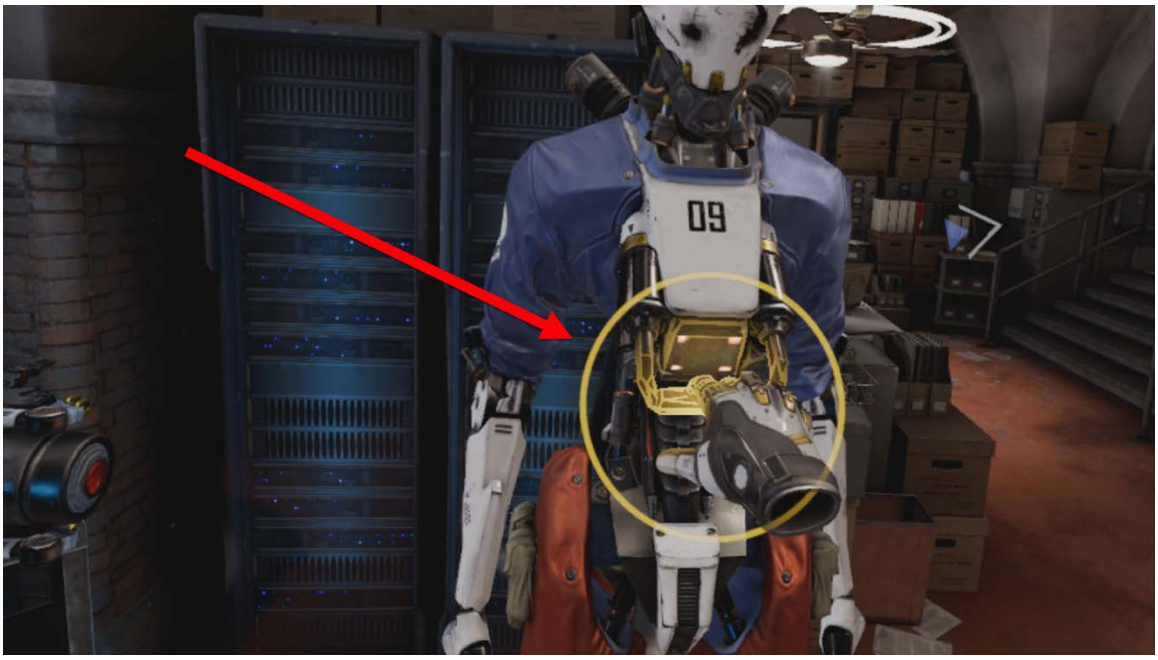
This is the warning given to players when they need to turn around for better tracking.

2. UI Elements

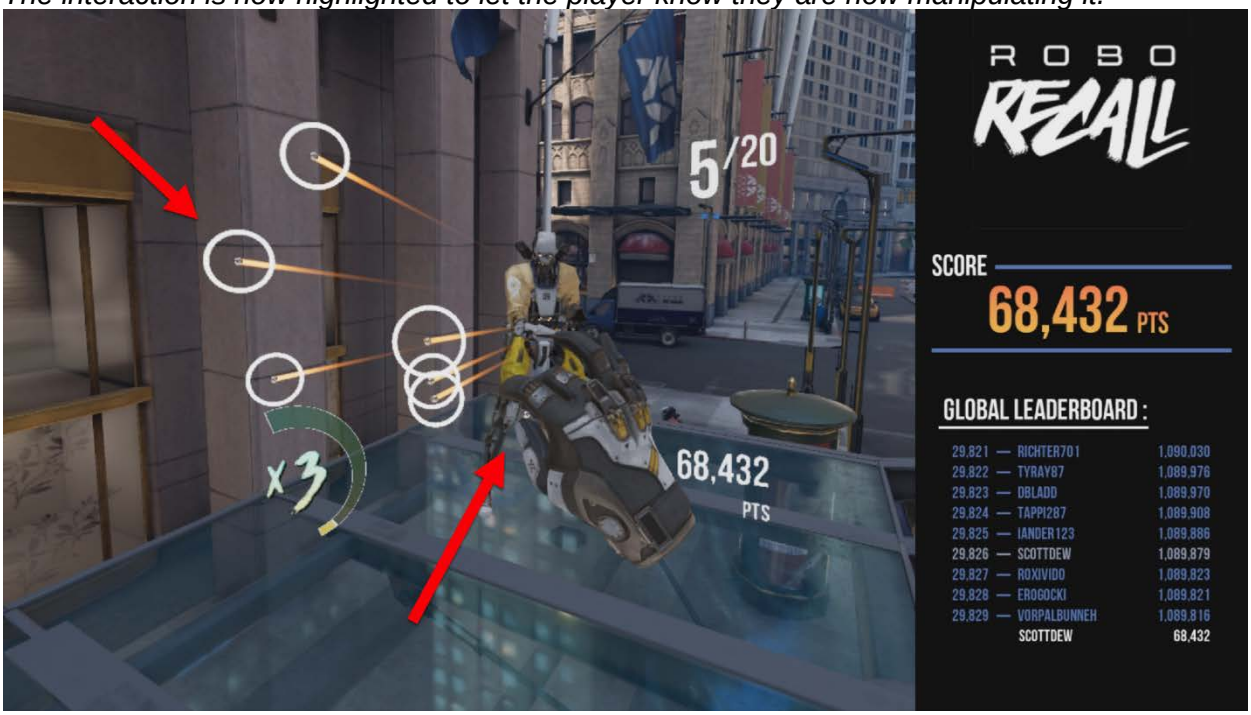
The UI in Robo Recall is straight forward, but one of the nice little touches is the tiny blue arrow on the sides of the screen, which point you towards an objective. It's subtle enough not to be in the way, but apparent enough to help you get where you're going.

The other UI Element that stands out are the little rings that appear around things to interact with. They do not become apparent until the player is close. There's nothing special other than that! Simple and clean. When the player interacts, the visual style enhances itself some. The ring will grow and part of the model becomes highlighted.

On the left is the circle to signify that the player can grab this item. And on the right, the little blue arrow telling the player that the objective is in that direction.



The interaction is now highlighted to let the player know they are now manipulating it.

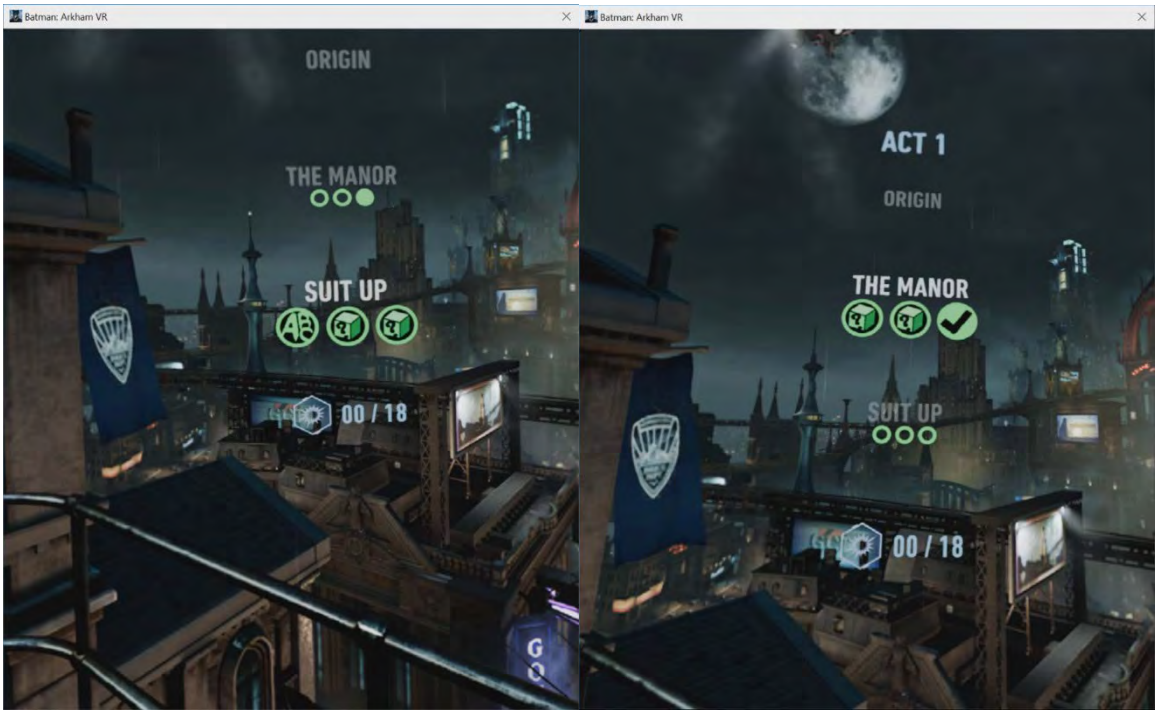


Once in the chaos of recalling robots, the circle interaction element can be seen with gun fire. This tells the player quickly that they can catch bullets and throw them back! And as fast as this game plays, this kind of language is extremely important to communicate quickly.

Batman Arkham Knight VR – Rocksteady Games

Story: Batman must find out where the toxin from the Joker Virus is coming from.
Visual Experience: Hyper Realistic / Photo Realistic
Key Design Elements:

1. **Look, Don't Point.**
   All the menus in the game will activate when looked upon. This even goes into the teleporting which I'll cover in a second. Most early VR apps, especially on cardboard, took this approach for menu systems. And you know what… it works well! There are no cursor icons to let the player know what menu they are about to select, other than the text and icons grow larger. The player just needs to hit a button on the control to select the menu/icon they want.



Notice how the text and icons are more noticeable when looking at each other? This is a clear signification to the player that this is the menu they can interact with.

2. **Display Elements**
   Interactive Elements are a little subtler compared to Robo Recall. Although once pronounced they use a more detailed graphic to showcase the interaction. And this works a little more for Batman, as it is trying to be as realistic as possible.

   And instead of letting the player just go where ever they want to, the game sets up specific hotspots that the player can teleport to and from. This gives a good illusion of being able to move around a VR Space, but also limits the scope and story for the player without them noticing. Which is fantastic if the need arises to make sure that players are only seeing what you want them to see, or what is in the exact scope of the story.

At the bottom is the icon showing which button to push on the controller to pick up the postcard. There is no other indication that the postcard can be picked up. It only appears when the user's hand comes close to the postcard. The Icon is a good indication to users, who might be unfamiliar with VR, to push and hold that trigger. In the back, we can see the Batman Logo with an A. If the user pushes the A button, it will trigger them to teleport to that location.

3. **Audio**
   It should come to no surprise that a heavily story driven game such as Batman is going to have a lot of audio. And this is going to be difficult explaining this in a text document…

   Sure we have Alfred talking to us, helping move the story along. And then there is Batman talking to himself (the player), which helps guides him around specific scenarios. But audio is also used in a very clever way in this game, especially early on. (I won't go into spoilers for the rest of the game, but it is seriously something to experience!)

   At the beginning of the game, Alfred tells Bruce that there is an issue he must take care of "downstairs". He hands him a key to the piano, and says he should check if the piano is in tune. At this point it is up to the player to either tune the piano, or check out Bruce's mansion a little bit. There is a red phone next to the piano, which the user can pick up! It will start to play Bruce's voicemail. The cool thing is, if the player wants to hear the what's on the phone, the user must put it up to their ear like a real phone! (See Below)

With the phone up to my ear, I could hear the voicemail. With it pushed away, it sounded very muffled!

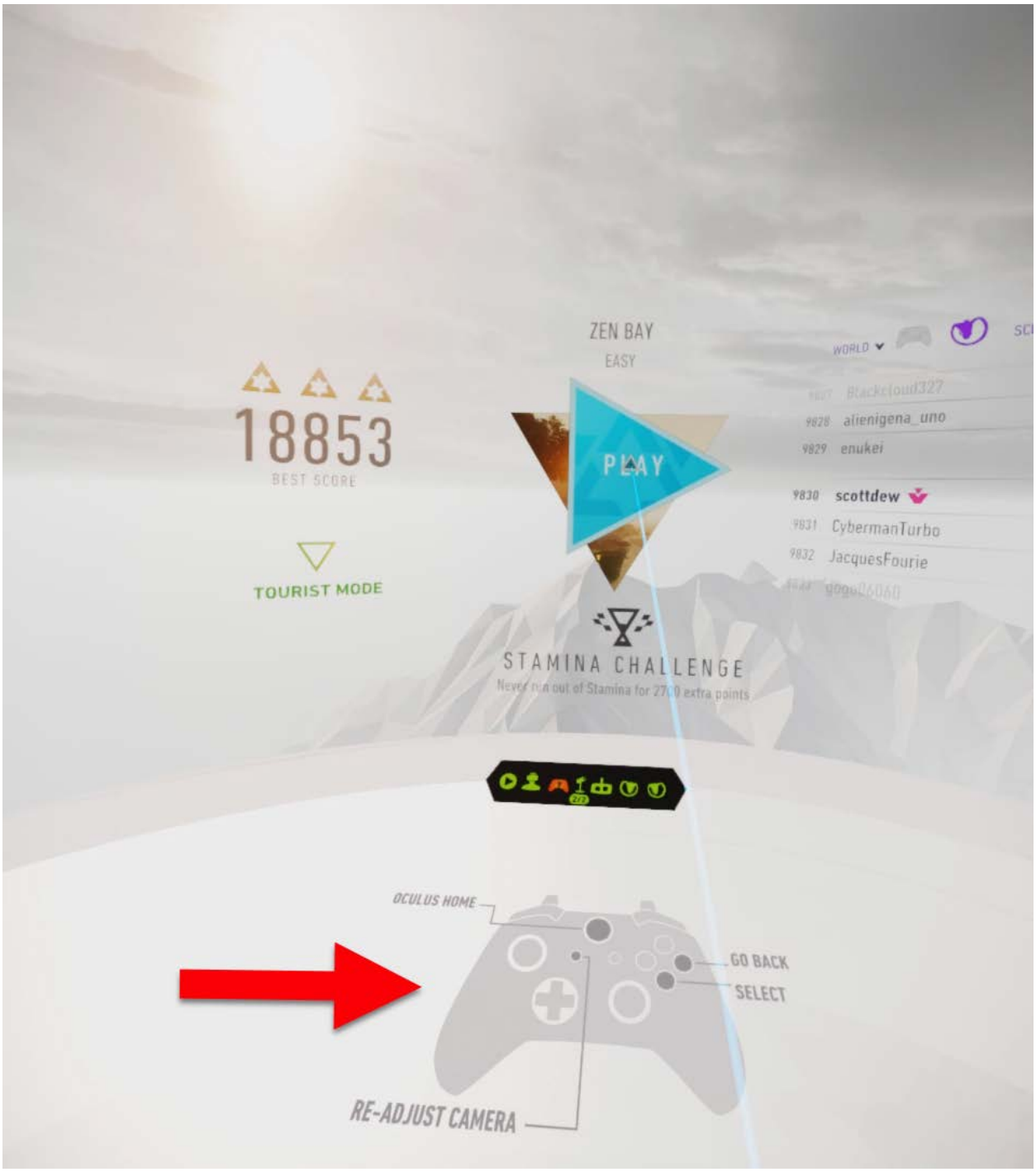
Now is this critical to the game, not entirely… But it sure does help ground the user in a world which isn't real! Now the other cool part about this phone is: If the player takes too long to tune the piano, it will start to ring! When the player picks up the phone, it is Alfred *encouraging* Bruce to come downstairs and to tune the piano. It's a nice touch to help nudge players along the story, instead of doing with a UI Graphic or HUD Element.

The Climb – Crytek Studios

Story: You like climbing mountains? Good. That's what you're going to be doing!
Visual Experience: Hyper Realistic
Key Design Elements:

1. **Graphic Design**
   The over-all graphic design in this game is top notch. There looks to be an incredible amount of detail in though into how all of it comes together. Everything is based off a triangle shape, which is the same typical shape of a mountain. All the UI Elements are Triangle Shaped. The over-all menu, and even the logo for the game, is shaped as a triangle! This kind of branding in an experience is hard to over-look, and should be considered as a highly important element. Especially if that experience is being design for a specific company and/or product. It just sells the package even more. This game's story is almost non-existing, so the over-all attention to the design helps compensate for the lack of story.

See the mountain? And I don't mean the one in the background.



Another nice touch is the control scheme at the bottom to help remind new users. Only downside is I wasn't using an X-Box controller for this experience.

2. **Wayfinding Graphics**
   The Climb takes a page out of the book for having “wayfinding that is not in your face at all”. There are no floating UI elements, no glowing objects, and no voice-overs telling the player where to go. The actual wayfinding is blended right into the environment, which makes it even more believable that the locations are real, and people do climb these cliffs for the adrenaline. The ledges that need to be grabbed all have a white chalky outline to them. This is the signal to the player to inform them of the path to take.

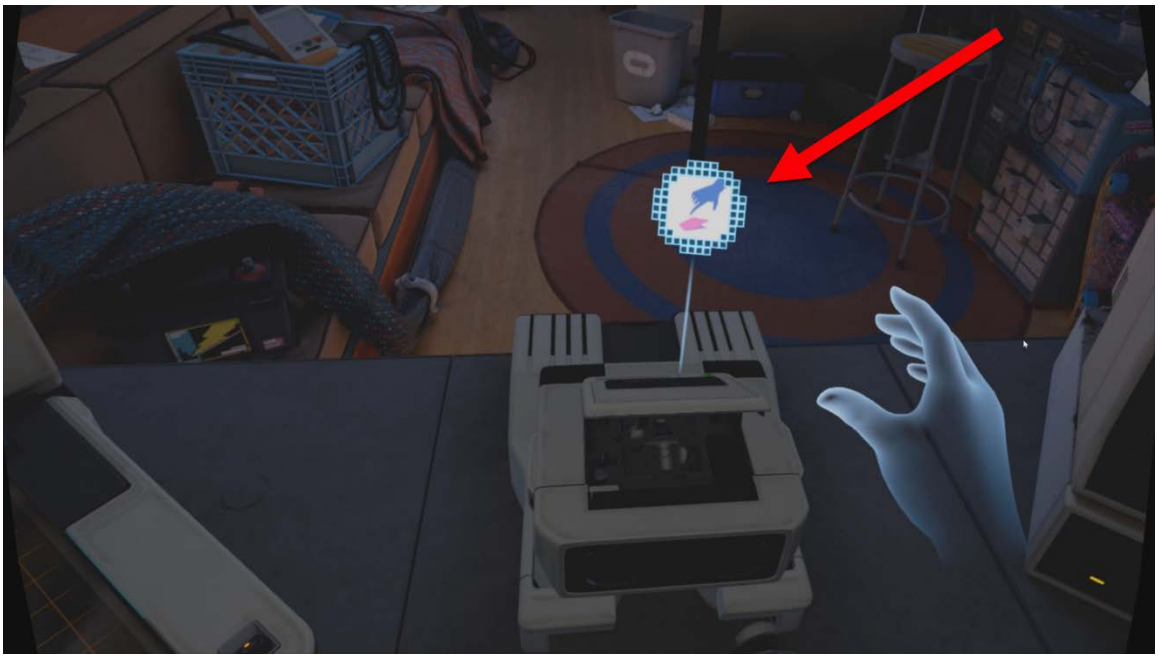
Which way do I go?

Even though there is an arrow here telling the user which way to go, at least it's painted onto the wall of the canyon. As if someone specifically came before player to help guide them through this adventure. Also, another nice tough is the check-point. It is a rope harness which when passed, will remember your location if the player falls. This could have been done with some floating flag or UI graphic. Instead, the harness keeps in line with the realistic design of the game's graphic design.

First Contact - Oculus

Story: Time to make a new friend with a cute little robot!
Visual Experience: Photo-Realistic
Key Design Elements:

1. **Display Elements**
   The entire experience has a very retro feel to it, and has a theme of “blocks” or 8-bit graphics throughout the entire design. Much like the Triangle design of “The Climb”. This is entirely apparent in the Display Element Graphics that appear to help the player do certain interactions inside of the First Contact.



Just push it down! But notice the blocky 8-bit look to the UI graphic. (It is also animated, but I cannot show that in this document.)

This style of UI graphic can be seen from any interaction in the experience. Next up is taking a floppy disk and inserting it into a computer. The robot even helps the player out here and repeatedly points where to put the disk:

“It’s dangerous to go alone, take this!”

There is triple help going on right now! We have a Display Element of the 8-bit block graphic, we have a Display Element of a blue highlight around the floppy disk drive, and our new friend is pointing where to insert the disk! It might seem like over-kill, but this is the first real experience anyone tries when they calibrate their Oculus Rift. So, the mindset here is: this person might not know how to do anything in VR at all. Which is important when you’re designing to your target audience.

2. **Character**
   This little robot is fracking cute. And the experience takes full advantage of this fact. The character design helps progress the story, as the robot will try to bond with you very early. In fact, the first experience the player has is to wave “Hi” to him. This action will progress the story, as the Robot will not come out of hiding until the player waves to him!

He’s way in the back right now, super scared. But then he peaks around the corner to wave “Hi”. Now the player could just not wave “Hi”, but most people’s first instinct to this is to wave back! This will progress the story to the next section here:

Now he's got a little closer to the player, but is still a little afraid. He waves again, and then the player should wave "Hi" back one last time. This will get him to come all the way up to the player to start the rest of the experience.

His cuteness, and likability, plays into this interaction. If the player does nothing, the experience will not progress. But most players don't realize this as everyone just ends up waving "Hi" to the robot! This is done without any prompt, display element, or audio cue. It's just the natural reaction people have!

Good character design and interaction can seal the deal in VR!

3. **Something Iconic**
   Another good way to move the experience forward, is to use something that players will just inherently know will do something big. Now not to spoil anything on this experience, but at the end of it, the little robot hands the player a gold disc with an Oculus Logo.

At this point in the experience, the player has no idea what is going to happen next. But the fact the Robot just handed the player something that **A)** Doesn't look like anything else so far in the experience **B)** Is shiny gold and **C)** Has the logo for the developer of the experience…. It's safe to assume at this point that this disc is special.

There is only on way to figure out what happens next!

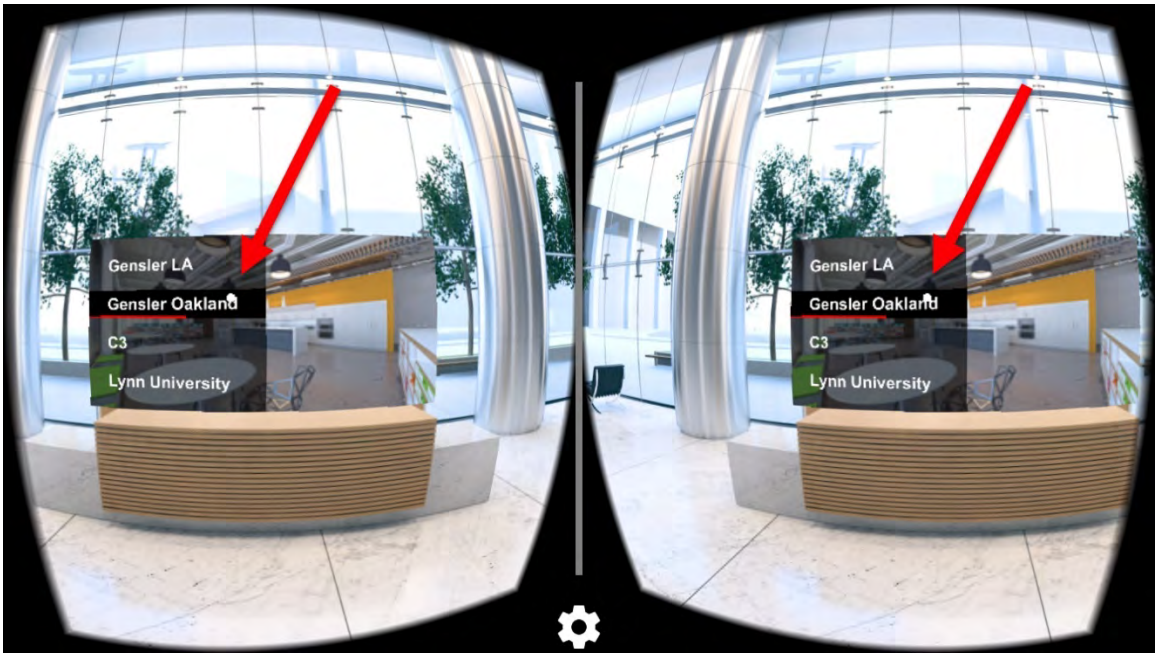
AUTODESK UNIVERSITY

GenslerVR – Gensler

Story: Up to our Designers on what story they want to tell with 360 Renderings
Visual Experience: Conceptual to Photo-Realism
Key Design Elements:

1. **Gaze Interaction**
   Since there are no real physical controls to go with the GenslerVR App, we designed everything to work with gazing. The user just needs to stare at something they want to select for roughly 3-5 seconds and it will select that menu item.

   Our main goal behind this was to make it as flexible on as many platforms as possible. We did not want to limit our application to one specific platform, such as GearVR. Although there is absolutely nothing wrong with picking a specific platform, it is just our client base is large. And having a single platform would be extremely limiting for us.

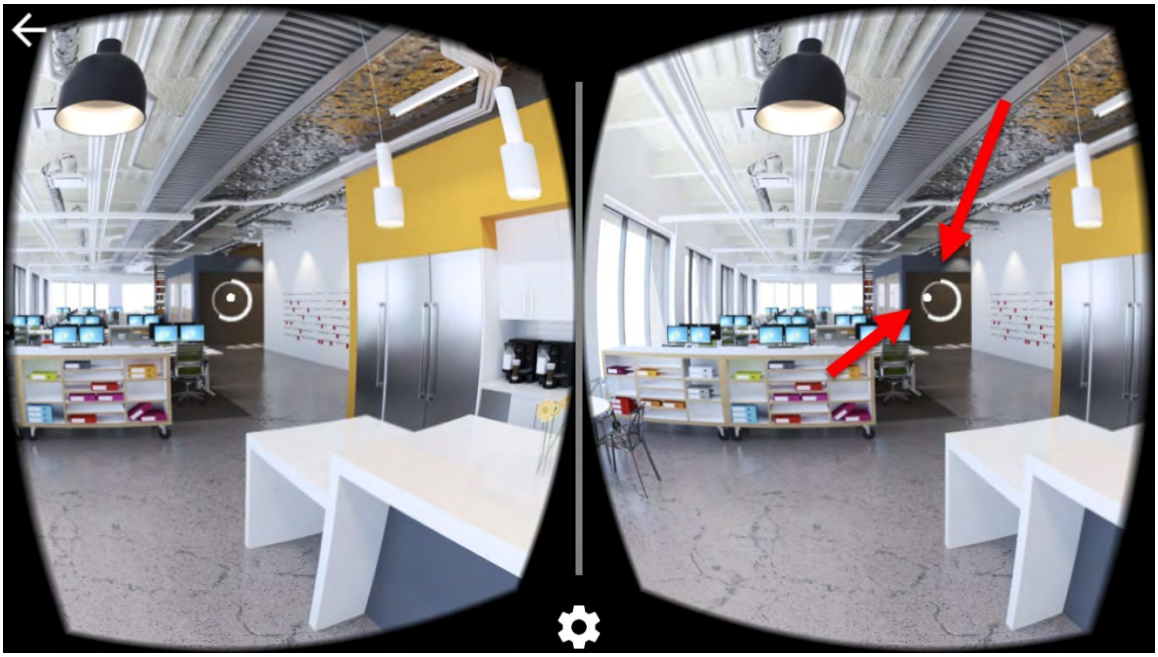


On the Main Menu, just put the dot on the Project you wish to view. Once the red bar moves all the way across the Project will load!

2. **Display Elements**
   Along with the simple navigation, we also opted for extremely simple graphics. We want the renderings and the design to be upfront, and not have any UI getting in the way.

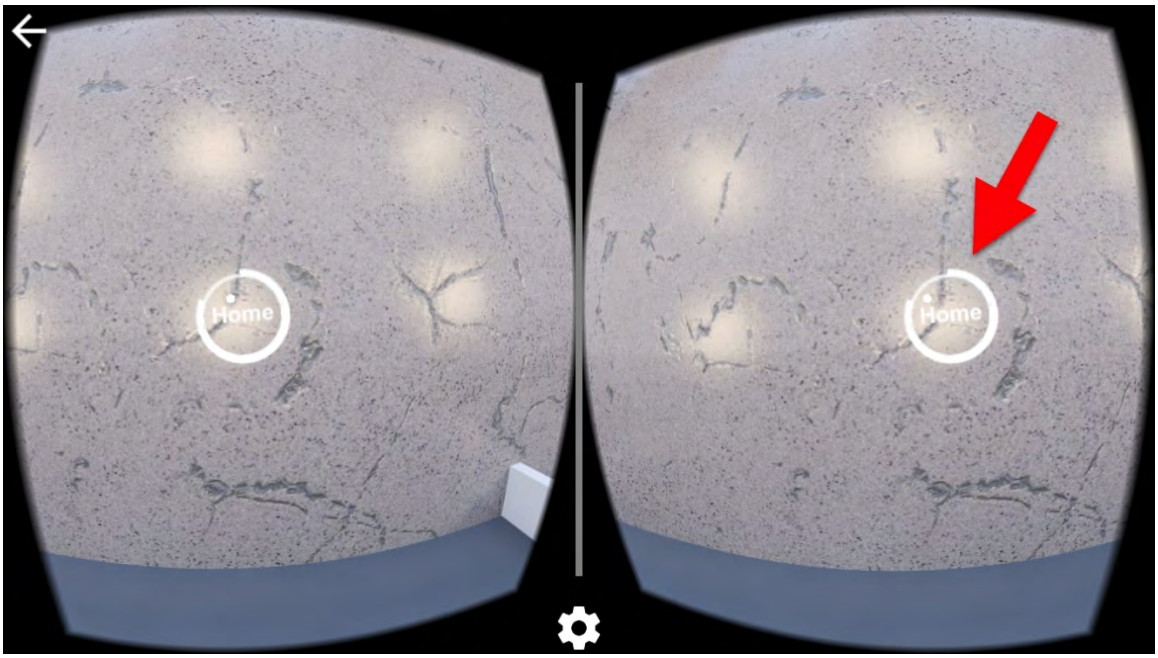
   We implemented a circular design for the UI for this, as our main cursor could fit easily within a circle. We also have the UI Fade in and out depending on how close the main circle is to a UI Element.

Notice the two little target circles in each eye, then notice that there is no circle over near the blue wall in the back….



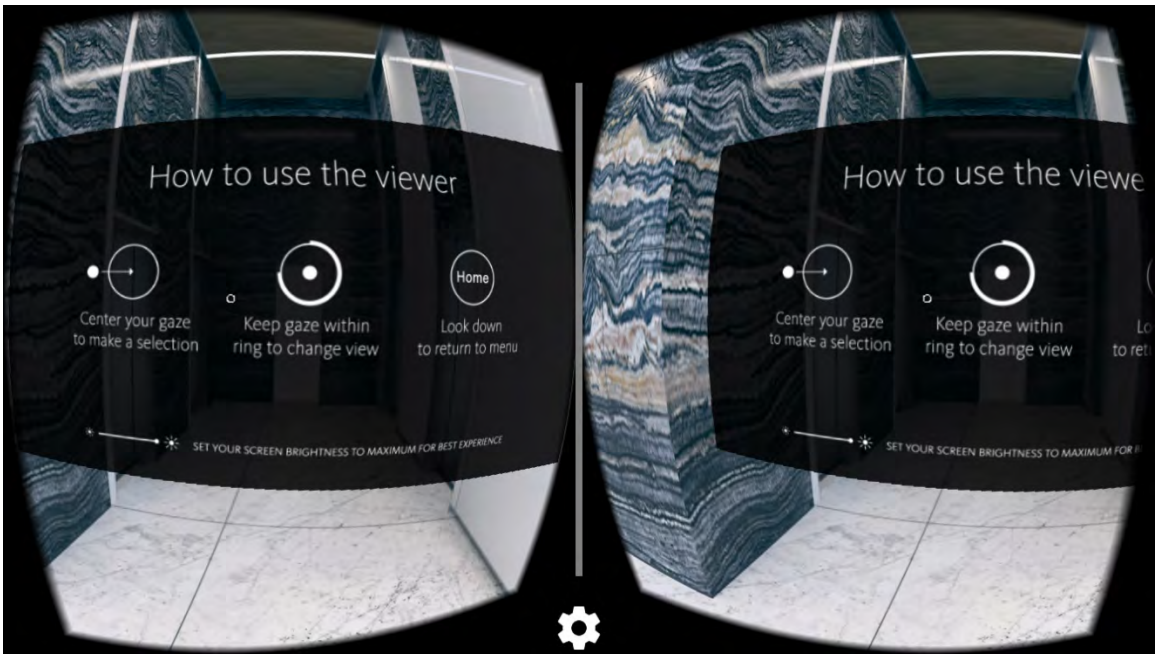
Now notice that when the small target circle is near a UI Element it appears. We even added a loading animation around the larger circle. This indicates to the user something is about to happen with this element. When the loading circle completes, it will move the user to another point in the project.

We took this one step further and added a circle to everyone's feet in the project. We found this to be the most natural way to move back to the main menu, as everyone always ends up looking for their feet in VR.



There's no place like home… (x3)

3. **Help Menu / Tutorial**
   One last thing we decided to add to the GenslerVR App was some brief instructions, in case it is someone's first time in VR. We landed upon putting the Help Screen in the Main Menu. We noticed the first thing people would always do their first time in the App was to look all around the Home Screen. (Instead of jumping right into a project.) This is what lead us to put the Help Menu behind the user, because when they looked around would find it pretty easily.



It is just that easy!

Applying to Practice

The best way to get started with all this information is right from the beginning. If you've started your VR Experience, it isn't too late. You may need to take a few steps back, and get ready to change a few things, so do be prepared for that.

The follow can prove as a possible example of how to approach the pre-production phase:
(I am not suggesting that the following is exactly what is needed for your project. That is up to you, the Designer, to determine. The following is just a possible scenario in which the suggested VR Experience would work well.)

Concept

The client is looking to sell a property. It's a $1.1 million home, and they want to use VR to apply to the selling process of the house. Delivery is for either a Rift of HTC Vive. The client is looking to have the VR Setup at their office, or at a site.

What kind of experience should we design?

A) The potential buyer is dropped into a VR model of the home. They can walk around and interact with potential objects such as: Turning the TV on and off, opening doors, or re-arranging the furniture?
B) The potential buyer gets out of their car at the end of a long day at work. They lock their car, and proceed inside through the front door. Once inside they notice a rose and a note on the table next to the front door. The note says, "Come and find me". A sound suddenly comes from the living room that sounds like the TV. The buyer walks down to the hall to the living room, and is prompted to turn the TV Off. Another sound of a kettle brewing on the stove comes from the kitchen. The buyer is prompted to turn off the kettle in the kitchen. After this, noises that sound like footsteps upstairs. The buyer is prompted to go upstairs. Once upstairs the buyer is hearing some noises coming from the bedroom. When the door is opened, there is no one in the bedroom. But then there is a glow coming from a "Home" Device. When clicked, the device says "I left something in the dresser for you…" The user moves over to the dresser, and finds a key and a note. The note says, "Check out the backyard". The user goes back downstairs to the back yard. Once in the backyard the user see's the grill on and with a bottle of wine next to it. There is another note on the table that says "Just wait till this is all yours!". And the experience fades into information about the house.

Which experience sounds better? We all end up doing version A quite a bit right now, which is why I wanted to create this class. We can do much better than A. Now B, might be a bit much, but it's here to prove the point of the experience. Something more than just walking around is a bit more exciting. Taking a page from some marketing, perhaps selling the product without focusing on it. While I was writing this, I had a commercial come on the TV for Pasta Sauce. But the commercial was completely focused on the family having dinner at the table. There was an occasional focus shift to sauce dripping on the table, or the plates sitting on the table. The point is, do not focus on just the technology. Tell a story. Make an experience.

Storyboarding

With the concept decided upon, it is time to block out the experience visually. This does not need to be anything super complex, but focus on creating enough visual information to create a roadmap for the experience. See the examples below for the experience we are designing from the concept:

Visual Experience

Now is the time to decide how this experience is going to look. Don't be afraid to spend some time on this. Gather a lot of inspiration from other experiences or art forms. For this experience, we'll be probably be going with a more hyper-realism look. This will allow us to get something close to photo-realism, but we can hide some performance issues within the art style. It will all be about compromises when going to VR in this aspect. But that doesn't mean things will need to be ugly!

Project Roadmap

Now our pre-production is complete, we can compile our Project Roadmap. Having a well-documented roadmap will help streamline the production and keep things on track. It will also help us realize our needed budget, and hopefully allow us to hit our target.

Models:

- Avatar Hands
- House
- Car
- Car Keys
- Surrounding Environment Neighborhood
- Trees, Plants and Grass
- Rose
- 1st Note
- TV
- TV Remote
- Couch
- Tables
- Books
- Additional nick-nacks
- Pillows
- Kitchen Table
- Kitchen Chairs
- Stove
- Kettle
- Refrigerator
- Additional Kitchen Appliances
- Bedroom Door
- Bed
- Dresser
- Side Tables
- "Home" Device

- Keys
- 2nd Note
- Patio Furniture
- Grill
- Wine
- Wine Glass

Audio:

- SFX: Key Jingle
- Audio: TV Noise
- SFX: Remote Click
- SFX: Kettle Brewing
- SFX: Kettle being set down
- SFX: Footsteps upstairs
- SFX: Sounds behind the door
- SFX: Door Opening
- Audio: Home Device Talking
- SFX: Dresser Opening/Close
- SFX: Sliding Door Opening
- SFX: Grill Sounds

UI:

- Start Screen
- Loading Screen
- Object Interaction Icons
- Ending Screen

Workflow and Pipeline

Digital Design Applications

There is a good chance all of us are starting with at least one of the following applications: Revit, Rhino, and Sketch-Up. Each of these applications handle 3D Geometry in a different way, and each will need to be cleaned-up in their own way. This is due to how the gaming engines handle geometry in their own way. The Real-Time Engines do not play with meshes that are poorly generated, or have too many polygons. It is going to become good practice for everyone to generate good clean meshes inside of a Digital Content Creator (DCC), such as 3ds Max. Excess polygons will increase the rendering time and power on the GPU, which will eventually lead to poor performance. There is no easy way around any of this, as these design applications were not created with Real-Time Engines in mind, or vise-versa. There are a few companies trying to solve this problem, and there are a few of them are starting to get there. But for these more complex applications such as Unity, Unreal and Stingray, you're going to have to roll your sleeves up a little bit and put some elbow grease into it.

This isn't the only area where optimization of the VR Experience will come into play, but it's a pretty big one. For this project, we will be moving everything to 3ds Max, but the same could be said for Maya or any other DCC application.

Rhino

It will most likely be best to export out Rhino geometry in either a DWG or OBJ File. Though if there is no polygon modeling in the Rhino File, an STL File is the preferred method going into 3ds Max. Either way, Rhino is going to give you some crazy looking geometry, and most likely need some good clean-up work.

Sketch-Up

As easy and versatile as Sketch-Up is, it can create some pretty funky geometry when exported. Thankfully 3ds Max will directly import a SKP file, so there is no need to export anything out. However, the Sketch-Up SDK hasn't been upgraded in 3ds Max for a few years, so you will probably need to save the SKP File to a 2015 format before importing into 3ds Max.

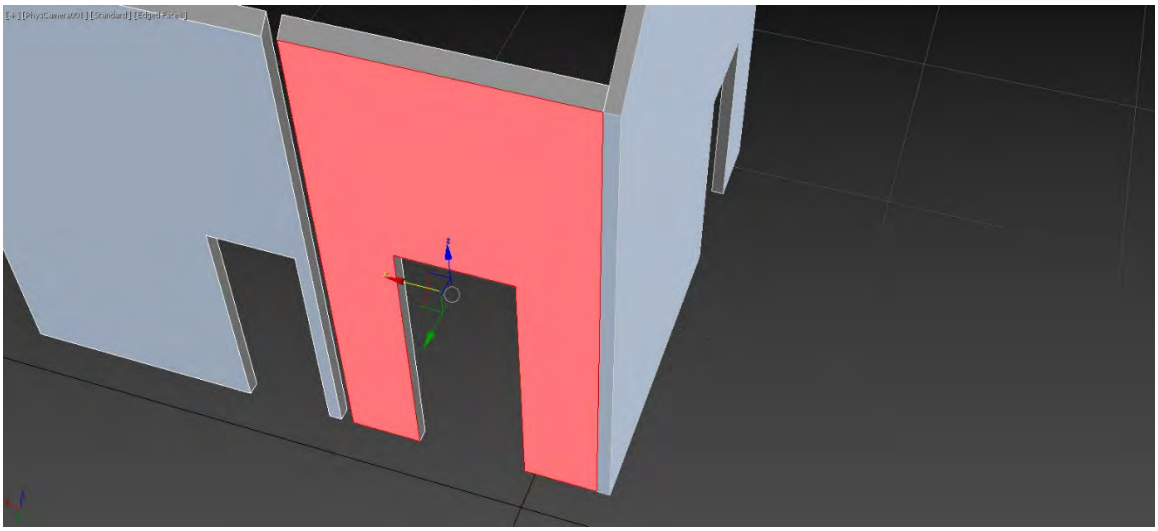
Also, watch out for "reversed faces" in Sketch-Up. If users don't pay close attention to this, there could potentially be a lot of clean-up work that will need to be completed prior to importing to 3ds Max.

Revit

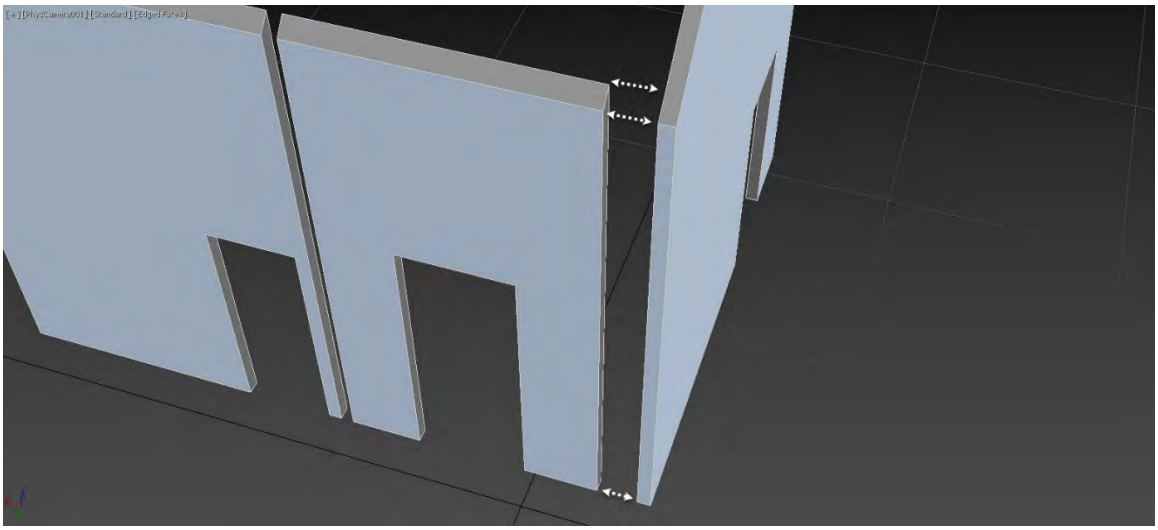
There are a few ways to get out of Revit. I have written a few AU Classes in the past about this, and I recommend reading this one here. The things that are not noted in this document for model clean-up will be associated with Walls, Ceilings and Soffits. Revit will create a lot of un-needed polygons when generating these objects. Anything you do not see should not be included in the exported Revit View. Yet, when modeling a wall, you will have essentially a cube (A 6-sided polygon). However, for optimization in the Real-Time Engines, there only needs to be 1 side of the wall. No one needs to see the other 6 sides. The additional 5 polygons are not needed. Users might be able to get away with this on some small projects, but it will become an issue on larger scale projects.

The second issue which will need to be fixed is with how walls are put next to each other in Revit. This won't look like an issue when inside of Revit, but it will become noticeable once the geometry moves into the Real-Time Engine. Revit will put the two walls right up next to each other. This will create a split in the walls where the corner turns around one of them, and will be located as deep as one of the walls. This is explained a little easier by the visual below.

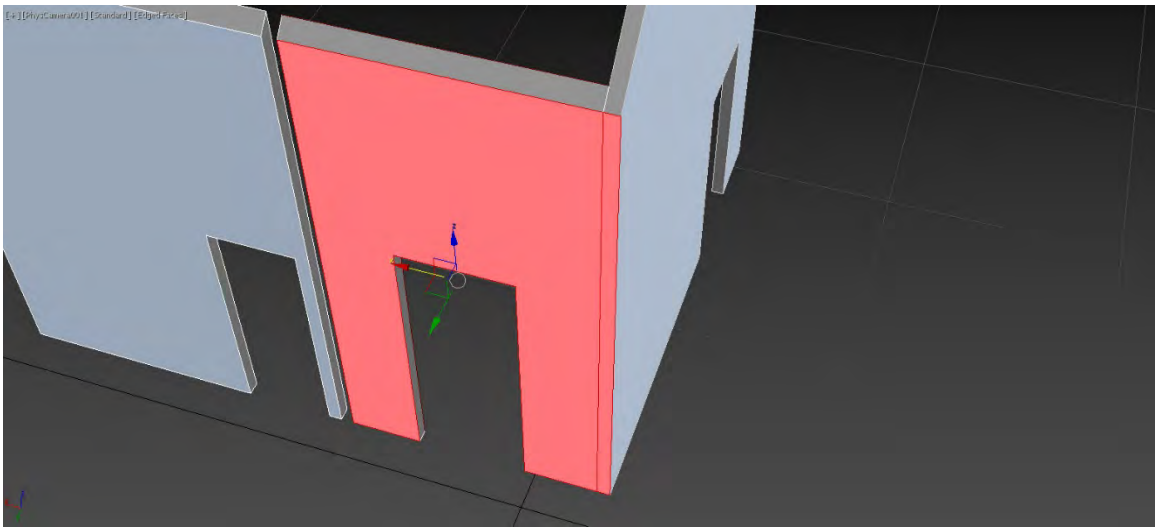
The problem here will be when users bake their lighting maps for the scene. Since these are two objects, this creates a split in the wall, and thus will cause a visible line of a shadow where the split happens. The effect is due to the light maps between the two objects not being synced up properly. There are two work arounds. The first is the recommended way, but will be the most work: The user will need to delete the excess polygons, and fill the left-over gap with a new polygon. The second way is to set the walls to Mitered, but in the end, you still have 5 additional polygons per wall. Although the split will happen at the corner of the two walls.



(This is taken from the Revit Sample House Project. This way you can inspect yourself!) Notice the highlighted polygon stops a little short of the corner. That is because this wall does not actually go all the way to the end. It will look correct in Revit, but Revit is lying to you! This is the problem.



Here I have moved the two walls apart from each other to show how it goes together.



Here I deleted the smaller polygon from the other wall, and then just extended the polygons from the other wall, on the left, to fill in the gap. This will result in the wall having a clean light bake.

3rd Party Applications

As stated earlier, there are a few solutions being produced that will help with the translations to a Real-Time Engine. Most of these are focused around Revit and 3ds Max.

Revit Live – Cloud solution for turning a Revit file into a Stingray (3ds Max Interactive) Project.

3ds Max Interactive – A solution to move a 3ds Max Scene to an 3ds Max Interactive (Stingray) Project.

VIMTrek – An enterprise cloud solution that will translate a Revit Model, with lights and textures, into a Unity Package File.

Epic Game's Datasmith – A solution to move a 3ds Max Scene to an Unreal Project. This will also translate V-Ray materials.

Umbra – Cloud solution for turning a Revit file into a Unity Asset. (In development at the time of writing this.)

3ds Max - Digital Content Creation (DCC)

For this class, we will be focusing on 3ds Max as our DCC**.** Yet, the topics here can be applied to any other DCC, such as Maya, XSI, Modo, or Blender.

3ds Max will be the Swiss-Army knife in the toolset for creating Virtual Reality Experiences. There are various toolsets within 3ds Max that can be utilized for moving to a Real-Time Engine, which might be new to some users who are not used to this kind of work. These tools are easy to learn and use. But those who are thinking this is going to be a super quick process, might be a little disappointed. There is going to be some work to be done here. Some of the work can be automated with 3ds Max Scripts or speed up with 3rd Party Plug-ins, but there usually is still some manual work that will need to be done.

Asset Management

Proper organization of the 3D Assets being used in the Real-Time engine will be key for a productive pipeline. Without any organization, it will become a painful process working with the assets inside of a Real-Time Engine, as these engines work in a “Modular” way. This means the Real-Time Engines view 3D Scenes as individual assets that will need to be re-compiled inside of the engine. It is important to view everything you are working for a VR Experience as an individual asset, and not an overall scene. For example, do not view the “House” in our example as an entire “House”. It needs to be viewed as what 3d Models make up the “House”. A few walls, windows, doors, etc. Each of these will become their own assets in the Real-Time Engine. The more a single asset can be re-used multiple times in the project, the better it is for the Real-Time Engine to handle the assets.

Naming Conventions

Having objects named “box3382” or “door01381”, will quickly become a problem, as the naming is too generic. Detailed naming conventions will help keep things tidy and organized. This not only goes for the 3D Geometry, but also the Material names applied to the 3d Geometry.

There are a lot of ways to handle Naming Conventions, and this will be up to personal preference and workflow. Or there is a chance the studio you are working for has set standards for naming that needs to be followed. Regardless, the names do need to be somewhat descriptive in nature. This will allow the assets to be easily found, and also easily inserted into any custom scripts, inside the engine.

Below are the guidelines I like to use for Naming Conventions. Not all of them need to be used, but they should follow the order given. Also, it is best practice to use “_” as a Space or Divider in the naming. This is good practice due to how scripting handles Spaces and other characters.

TYPE_MANUFACTURE_BRANDING_LOCATION_FUNCTION_STYLE/COLOR _LEFT/RIGHT_Animatined/Static_0000

Example:

DOOR_Front_Left_Animated_0001

CHAIR_HermanMiller_Aeron_OfficeSpace_Black_S_0001

Modeling and Re-topology (Mesh Clean-Up)

NOTE: Whole books, and hours of online training, have been created around this one topic alone. I'm not going to be able to cover everything under the sun that could be cover in this section. There could even be a whole dedicated AU Class for this section alone, which still probably wouldn't be enough. I am going to try to give enough of an over-view to point you down the correct path. There is a good level of 3ds Max, and 3D Modeling, understanding expected in this section. If you are brand new to 3ds Max and 3d Modeling, trust me when I say a simple Google (or Bing) search will yield and incredible amount of resources around this topic. So please use this as a starting guide to get you going!

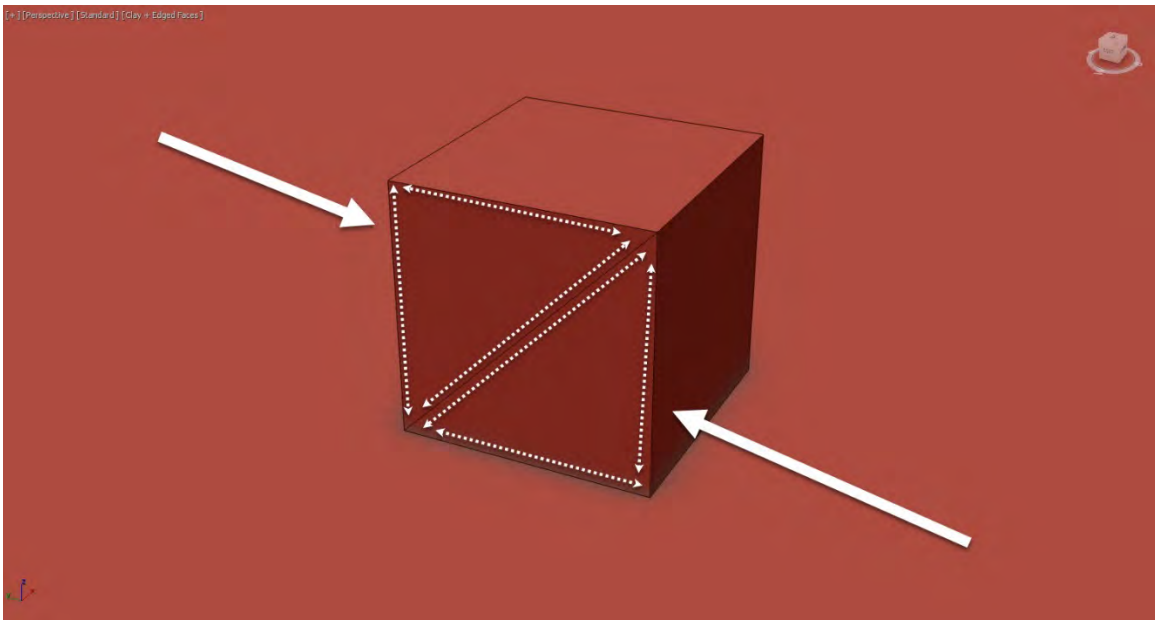
The main point of this phase is to make sure that the geometry is optimized for the GPU, as there can be a lot of information that needs to be displayed in the Real-Time Engine. And the more polygons that are on the screen, the longer the GPU will take to render the frame. Remember, we are aiming for 90 frames per second! Clean meshes will also help the next phase, which is UV Mapping/Unwrapping. It makes life so much easier all around, so it's super important not to skip this step, or try to speed through it.

There are certain guidelines to follow when cleaning up geometry. Sometimes these guidelines will need to be broken, but it is important to try to follow them as close as possible for the best possible results.

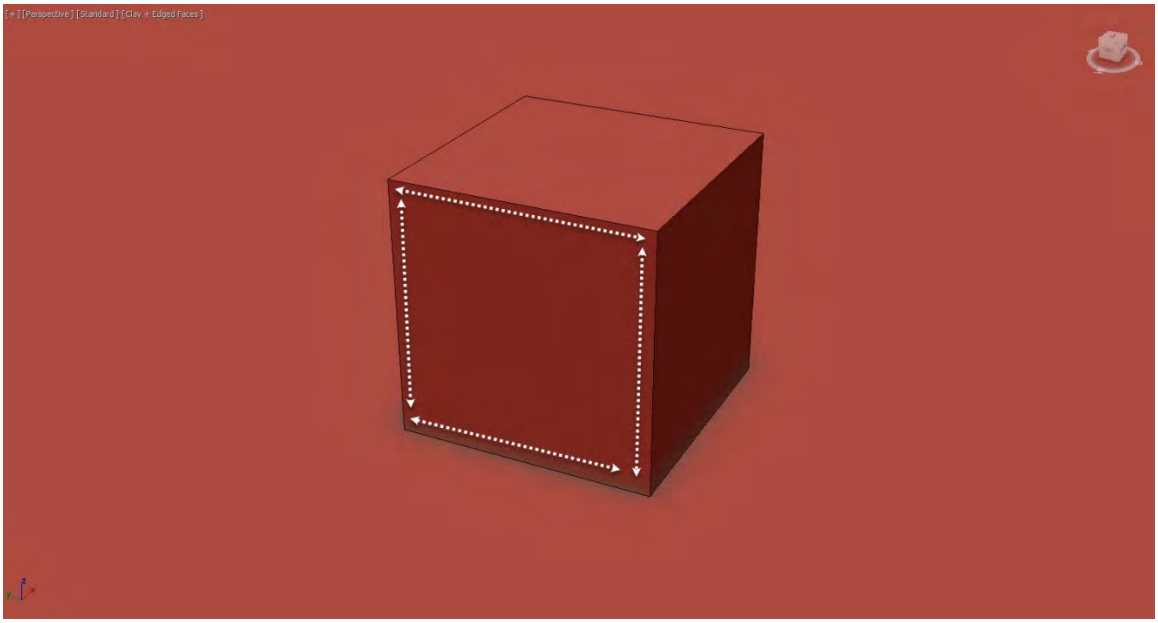
NOTE: I highly recommend using Edit Poly when working with meshes in 3ds Max.

Polygons (Quads) and N-Gons

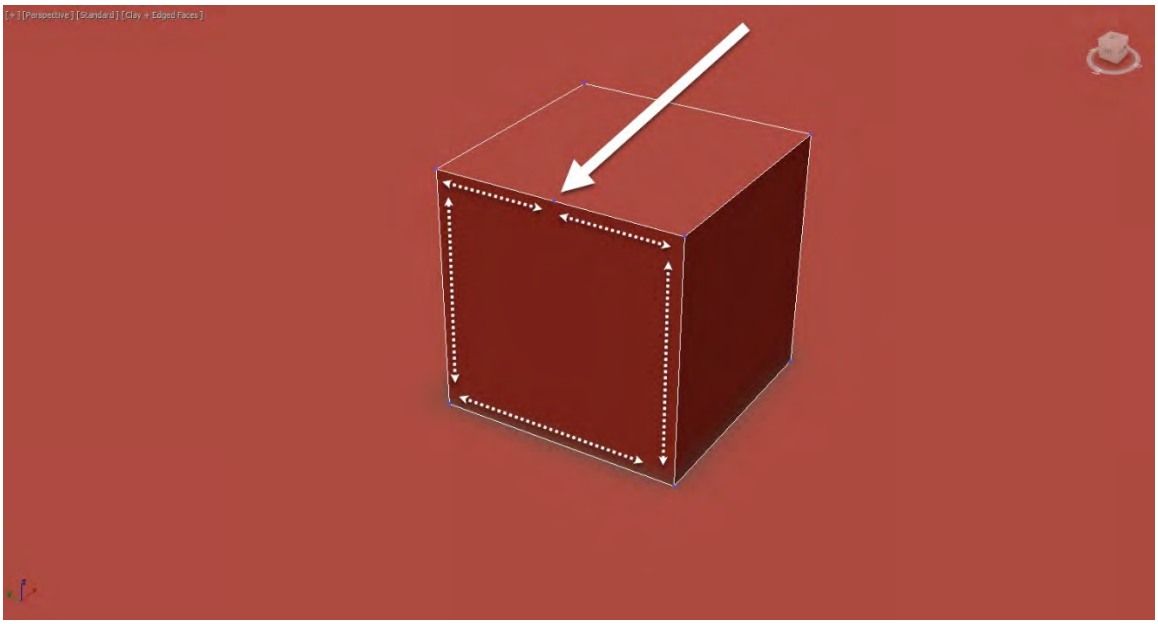
In 3D the basic building block is a Triangle. When two triangles are put together the create a Square, aka a Polygon. Optimal meshes are make completely out of Polygons. Polygons that contain more than 4 vertices are known as a N-Gon. The "N" is a place holder for how many edges the polygon has due to its extra vertices. N-Gons can create funky geometry in CG Applications, especially Real-Time Engines. If the object is static, they can be "ok" to have in the model. (But not recommended). However, they should be avoided at all costs in models that are animated.



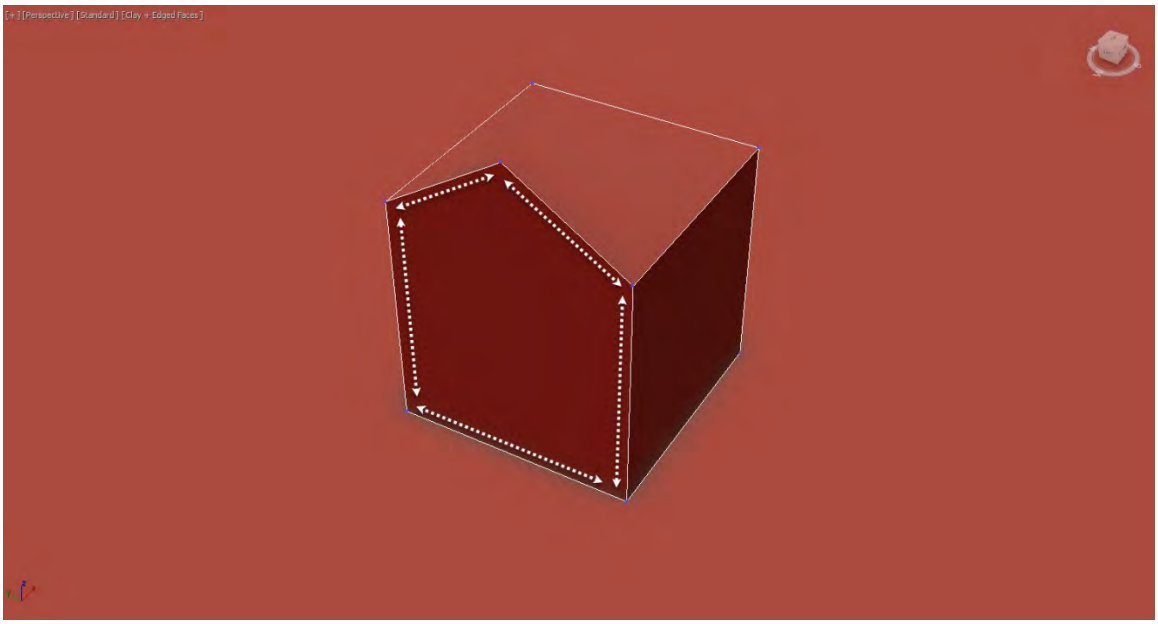
These are two Triangles forming a square, which is a Polygon.



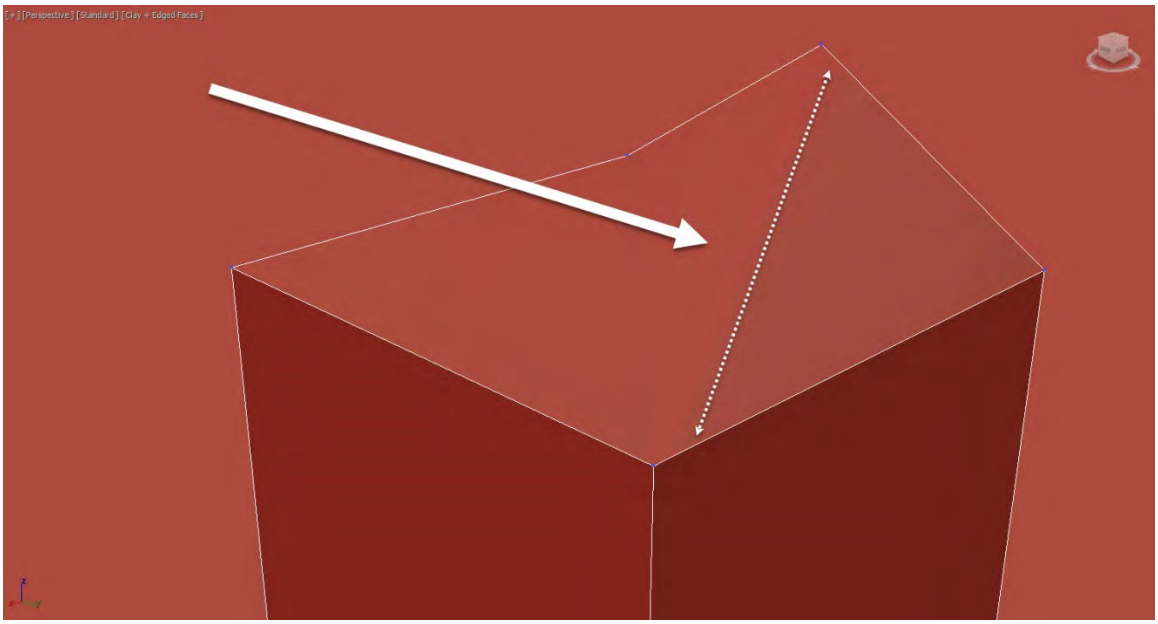
Just a Polygon does not have the edge down the middle, but it technically is still there!



Here we have introduced a single new vertex at the very top of the polygon. 3D Applications no longer see this as a Polygon but a N-Gon. Or in this case a Pentagon.



It is easier to see when we raise up the single vertex just a little bit. We also can see it's starting to do something funky to the top polygon, which is now also a N-Gon.

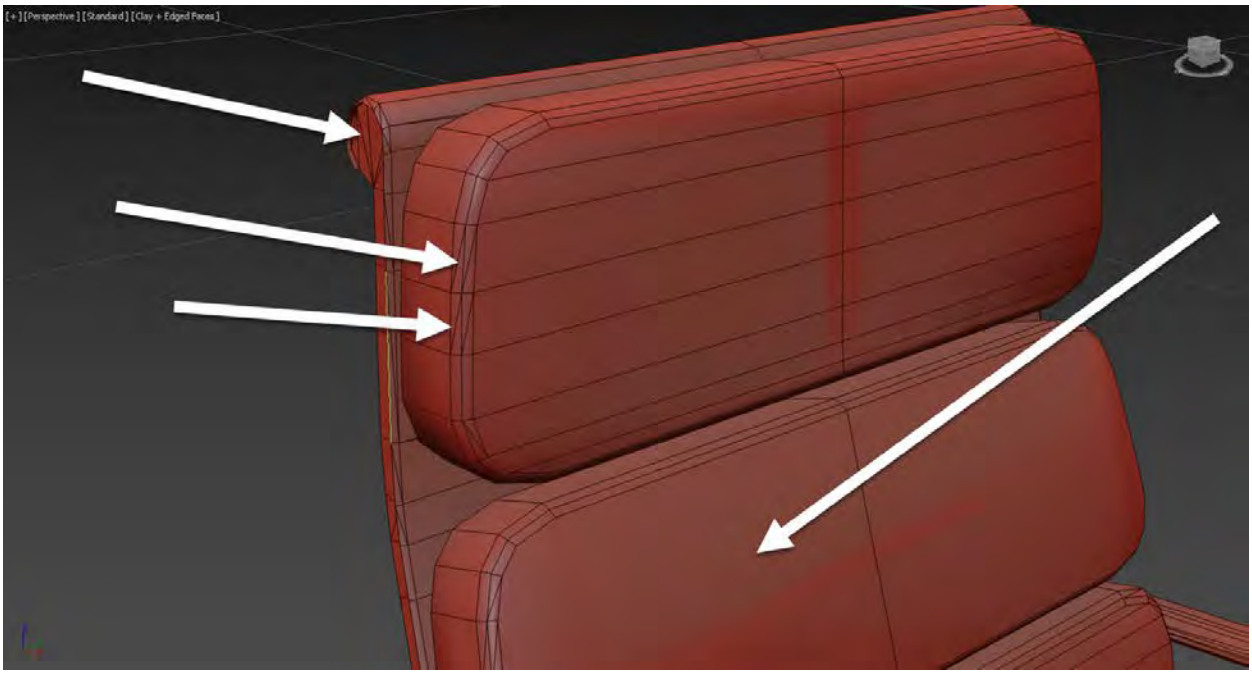


There is some weird distortion happening here. Even though this is an extremely simple example, imagine this happening on a more complex mesh that is being animated! Best to stay away from these problems if possible!

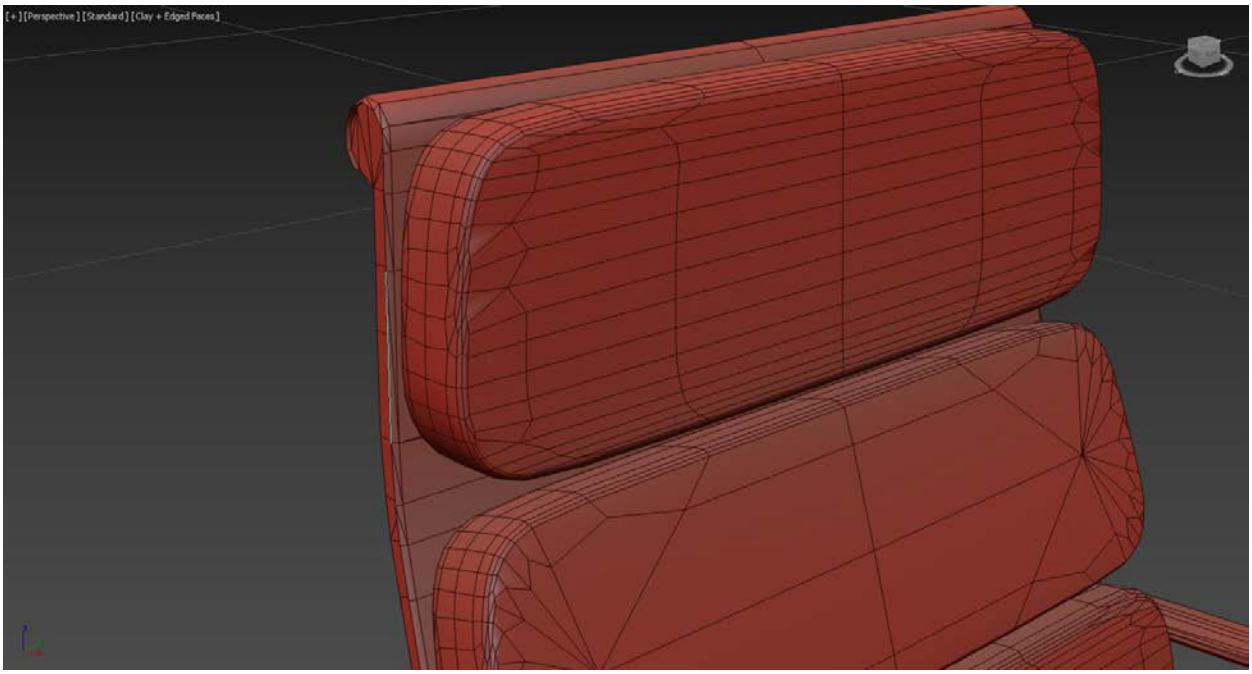
> **PRO TIP:** I recommend setting up the 3ds Max Quad Menus when users are becoming more comfortable when modeling. The Quad-Menus are an extremely powerful feature when working within 3ds Max, as any command/tool/script can be set up as a right-click shortcut in the view port! 3ds Max already has some key tools there, but customizing this to fit your style is extremely helpful. This can all be found under the Customize > User Interface > Quad Menu.

Topology (Edge Flow)

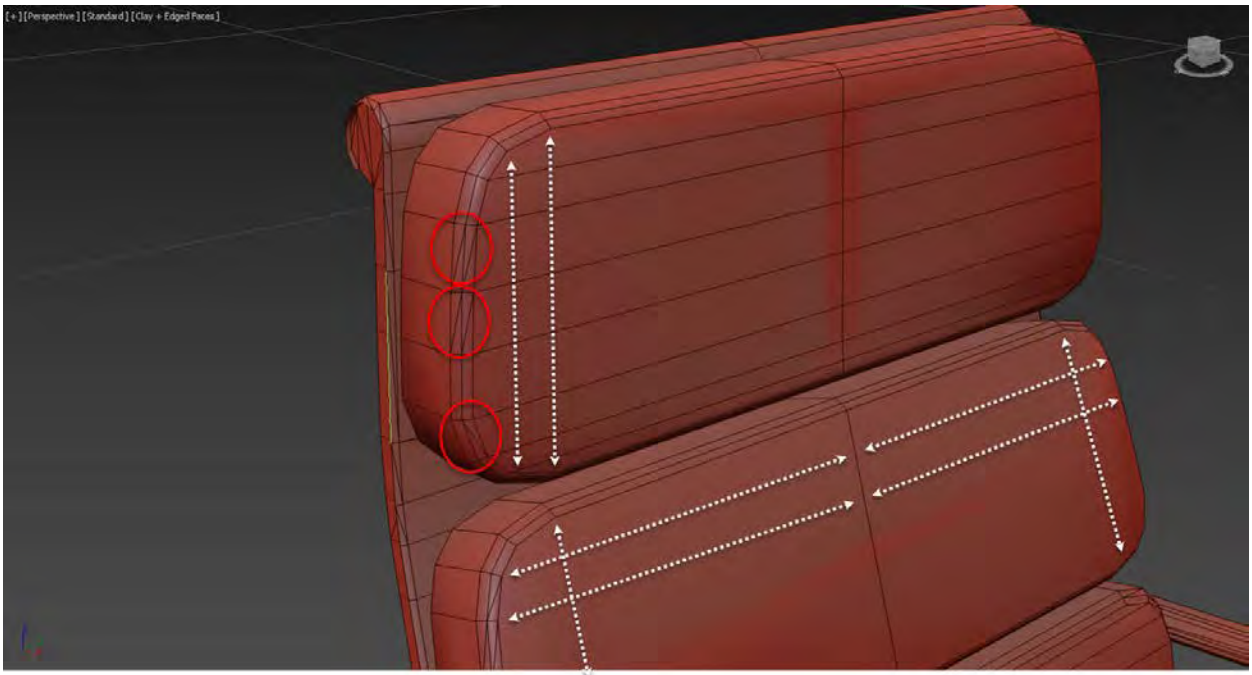
As polygons come together in a 3d Model, they create a natural language between each other. This is known as Topology or Edge Flow. It is extremely important to understand this, as “proper” Edge Flow or Topology will lead to a clean-model. This is both a science and an art. Topology becomes extremely important in Characters, Organic Modeling, or Complex Hard-Surface Modeling. In Characters, proper Topology will help immensely with Animation. This can be seen in just about any professional 3D model of a Character. The Edges Flow in certain direction around key areas, such as the eyes, the mouth, and joints. In Hard-Surface modeling, proper Topology will help create additional details in the model such as chamfered edges, insets, and extrusions. This Topology becomes even more important in the next two guidelines as well.



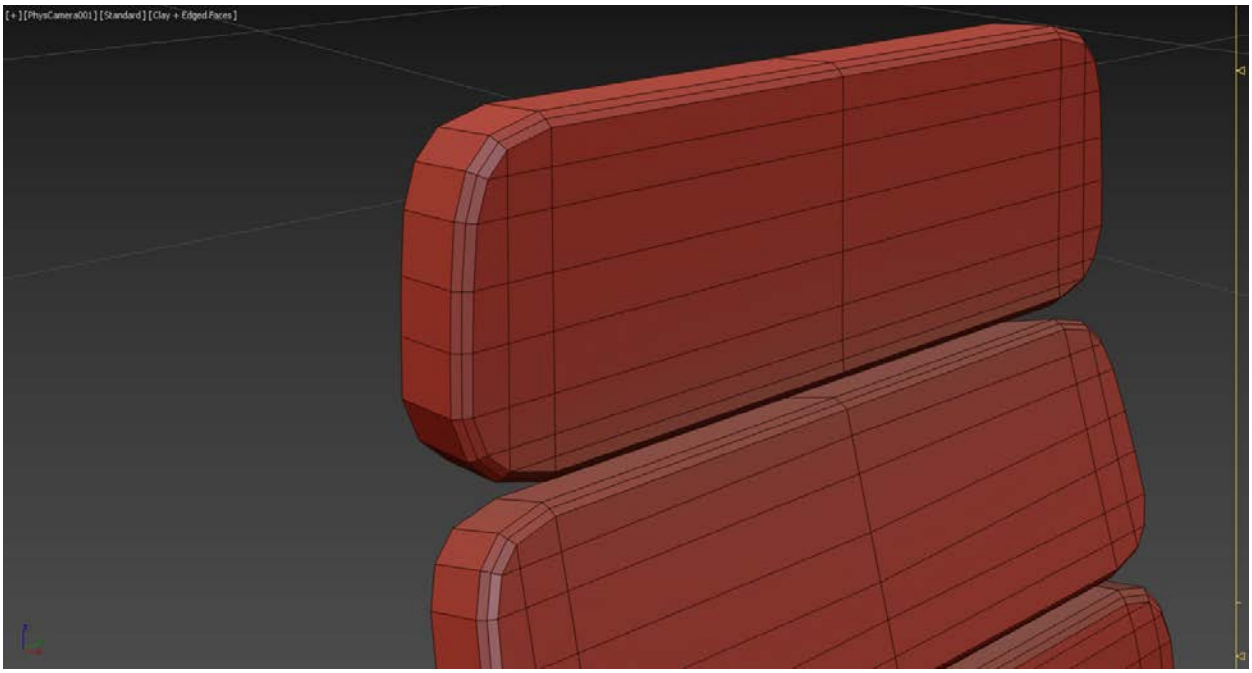
Alright, so this chair isn't perfect, but it is not terrible either. The biggest issues we have right now are some additional edges that do not need to be there (on the left), and then we have some missing edges on the seat cushions. These missing edges kill the proper flow of the edges for the chair's cushion. Let's put a Turbosmooth on to see what happens...



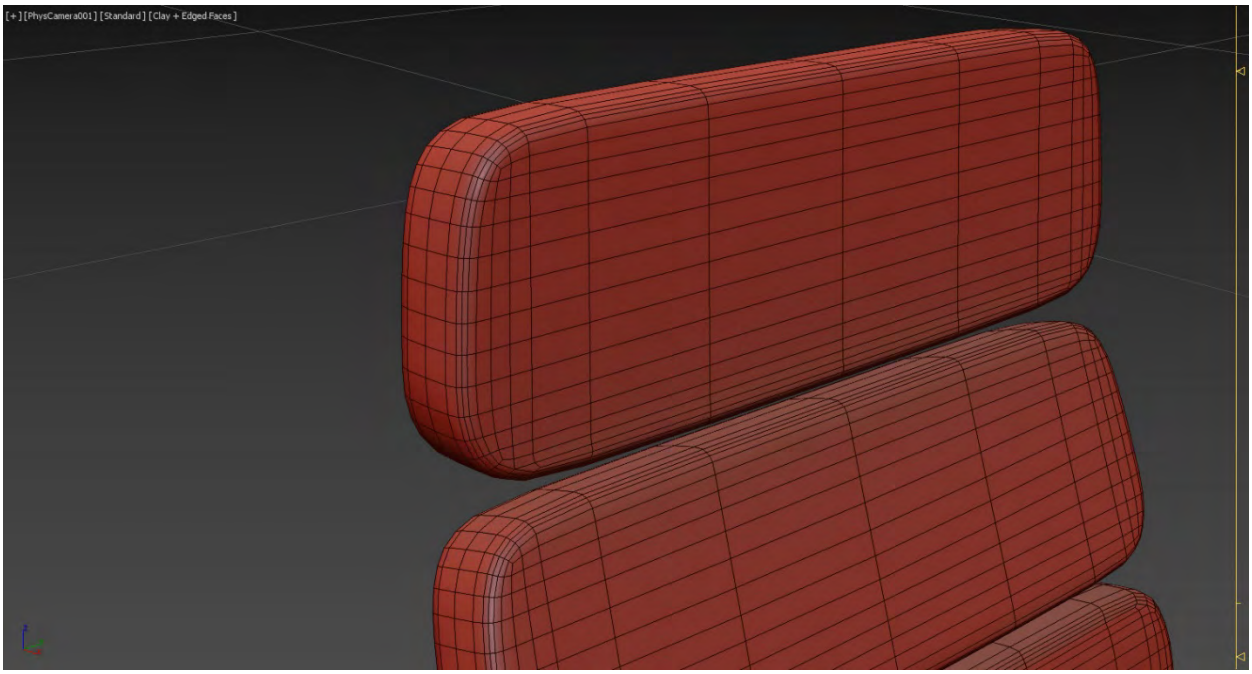
Yep. That's not pretty at all… Those missing edges are causing havoc on the cushions.



So, let's look at deleting these edges (the red circles), and adding in some additional edges to help with the edge flow…

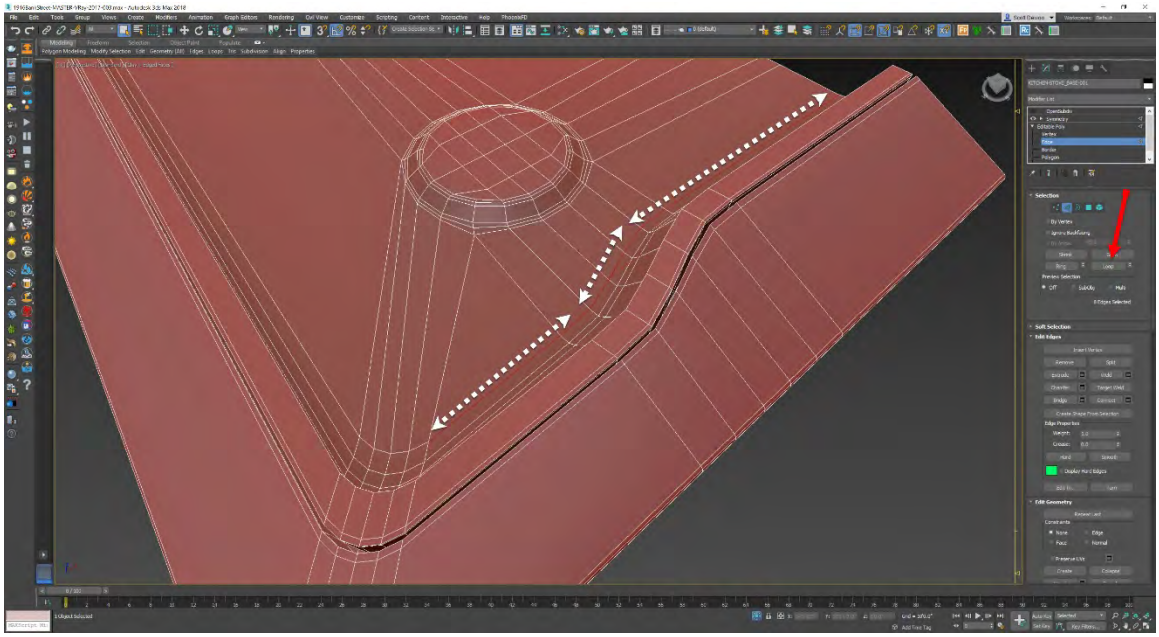


That looks much better! We have removed all the N-Gons from the object, and cleaned up any un-needed edges. Now when we turn on TurboSmooth…

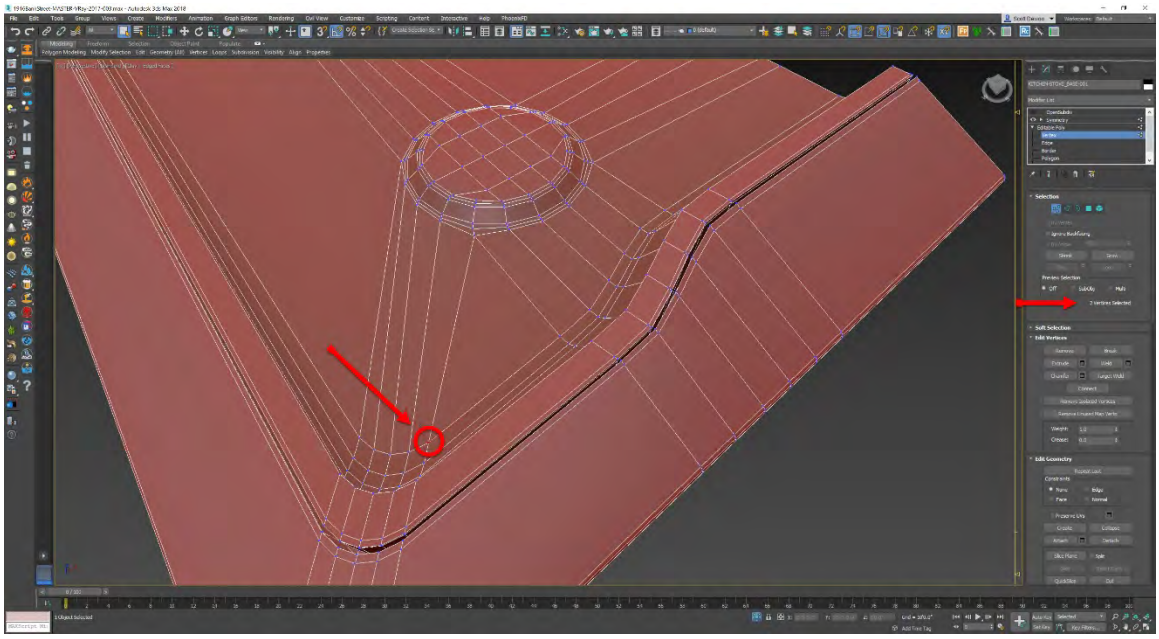


The cushions now deform properly, without any artifacts or nasty looking polygons.

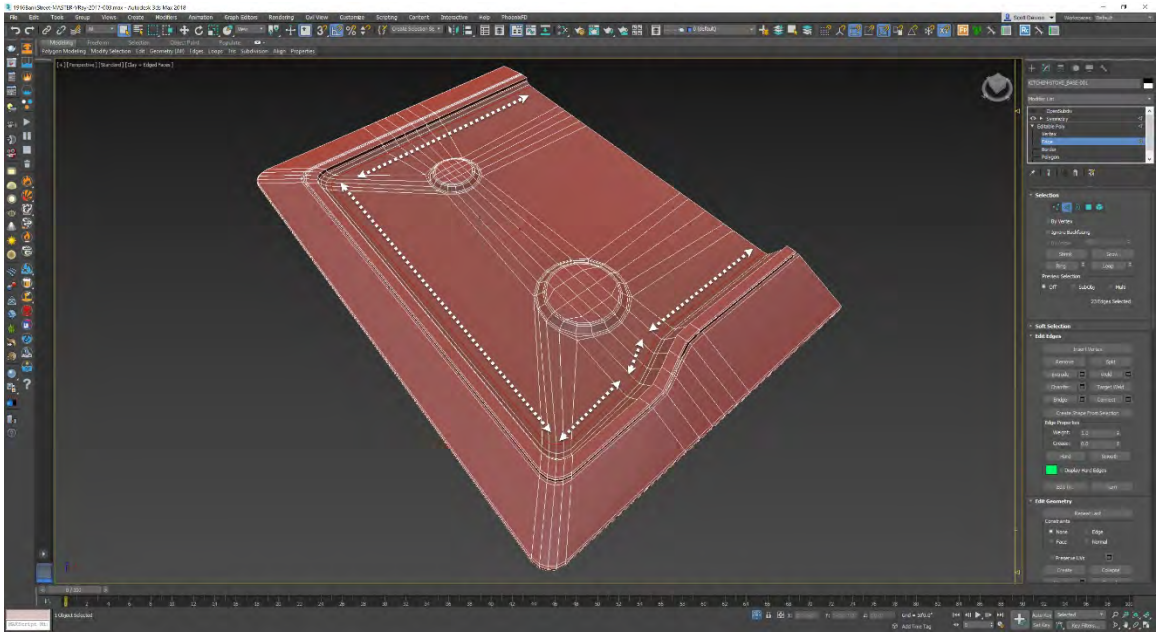
TIP: An easy way to check how your Topology is coming along for your 3d Model (Because there isn't a true exact way for every model), use the Ring and Loop tools inside of the Edit Poly Modifier. These will select all edges in a specific direction. If the selection ends abruptly, there is a problem at the end of the selection!



Selecting this Edge, and then clicking "Loop" under the Edit Poly will show that the edge doesn't continue when it should.



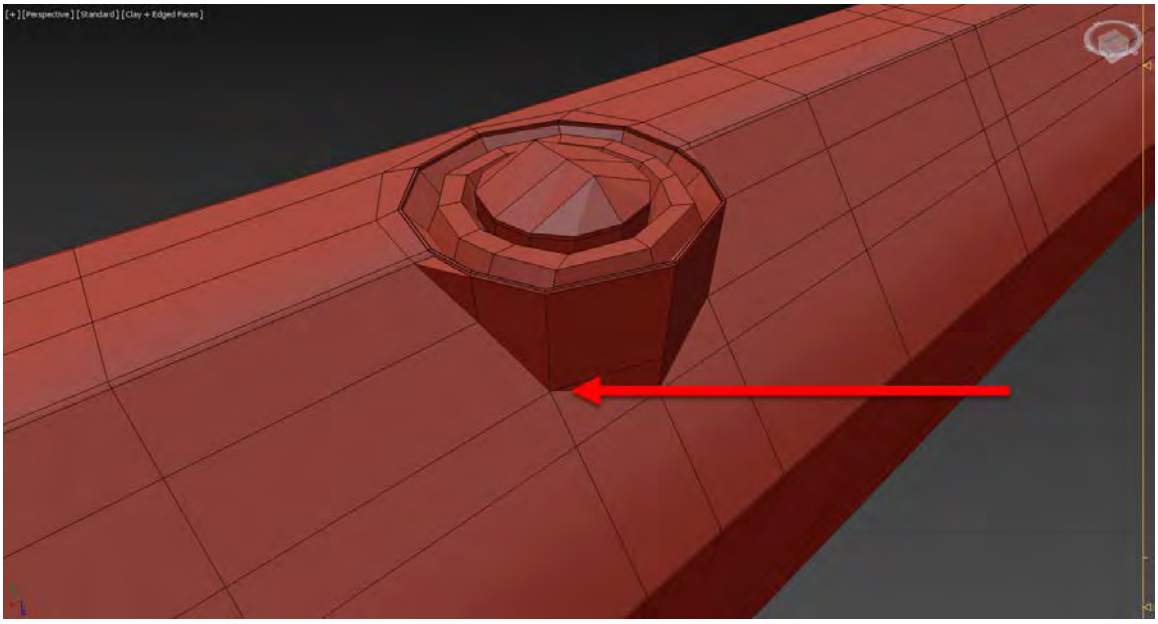
What we have here is two vertices that are not connected! So technically this is a gap in the model. If we used the Wield Tool to combine this to one vertex, it will fix this.



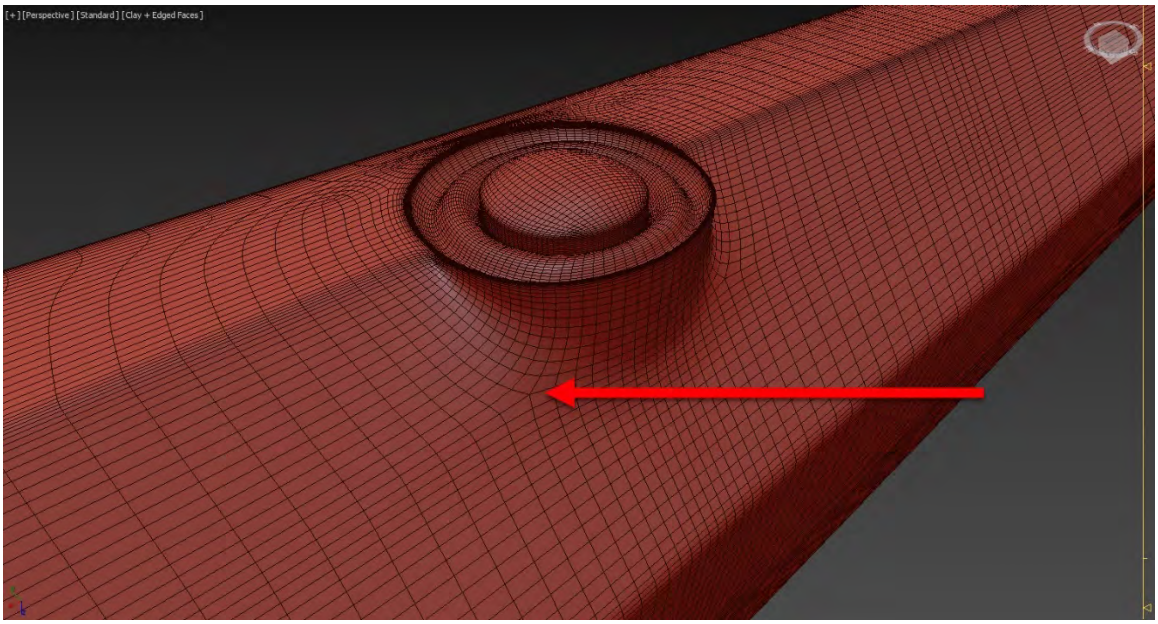
Now if we select an Edge and click “Loop” again, the selection will go all away around the model!

Avoid Stars

When 5 edges (or more) come together, they create what is known as a “Star”. This is due to the shape the edges make as they move away from a vertex. (Illustrated below). These are some of the worst artifacts that can happen in a 3D model, and can be a pain to remove sometimes due to how the Topology is working. They will create major issues with using Smoothing Groups or a Mesh Smoothing Modifier. And they will cause extremely funky looking results when they are animated or deformed. Users will most likely have to rework a lot of the Topology of the mesh to remove them, but the effort is worth it. Sometimes users will be able to get away with the Star, but those times are few and far between. This is the time that modeling becomes a puzzle, so take a step back and look at it from that perspective. Usually things will start to come together at this point… (No pun intended)



So oddly enough as I’m trying to write this document, I can’t find a working example of a Star… I did however find my mesh has a few stars that give me the effect that I was looking for when modeling.



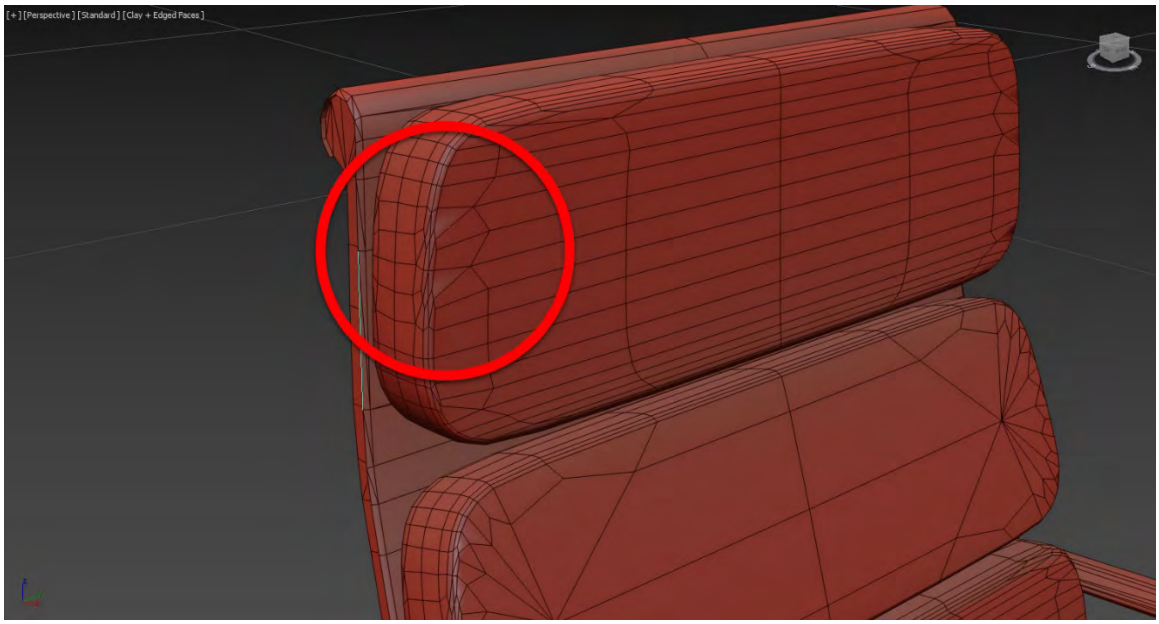
Here you can see it's still giving an odd deformation with the smoothing, but this object has this blended hard/smooth surface feature right here. Thankfully this object doesn't need to be animated. However, this will most likely cause some issues when baking the normal later. I should probably work on cleaning this up in the future! Avoid at all costs…

Mesh Gaps

When importing meshes from another application, there is a good chance there is are Gaps in the mesh. Even if Vertices and Edges look perfectly align, there is a chance they are not connected. This is what creates the Gap in the Mesh. This needs to be address as it can cause issues within the Real-Time Engine. Gaps can easily be cleaned up with the Wield, Target Wield, or Collapse Tool in the Edit Poly Modifier. Since these can be hidden, there are a few tricks to implement into examining an imported mesh:

Add a TurboSmooth Modifier

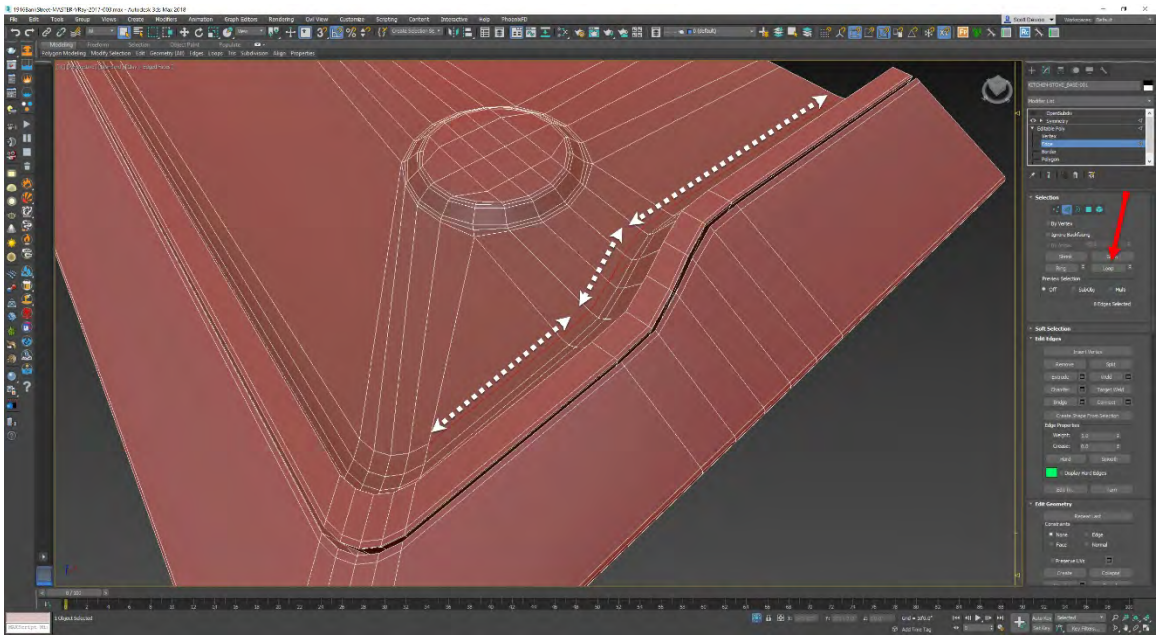
By adding a TurboSmooth Modifier to the Mesh, it will cause the mesh to be subdivided; leading the Topology in the area with the Gap to deform. So, any imperfection in the Topology of the 3d Mesh should be examined for a Gap, or any other potential issue!



Looking back at the chair from a previous example, here is the TurboSmoothed version of the pre-cleaned version. Notice the artifacts here on the side. This is due to the additional edges that we ended up removing.

Ring/Loop Tool (Edit Poly)

Just as I stated about checking for proper Edge Flow, using the Ring/Loop tool should expose a Gap when the selection doesn't continue as it should. This will most likely expose either a Star or a Gap, and should be fixed appropriately.

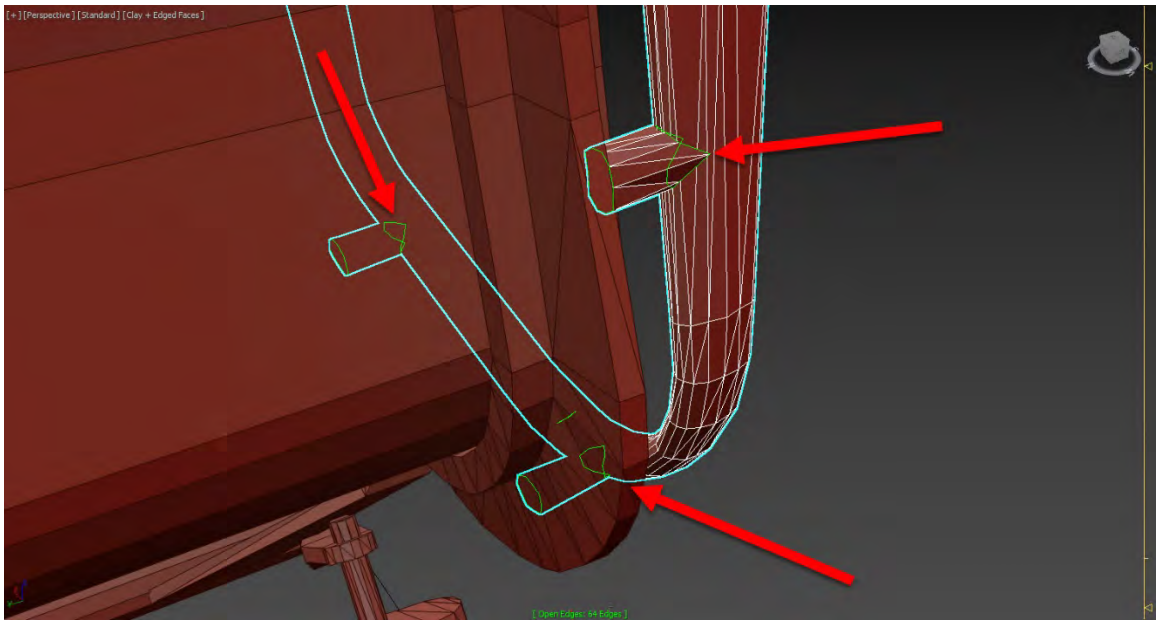


As in a previous example, we can see how using the Loop and Ring selection tools can show us where an issue in the topology can arise. In this example, the

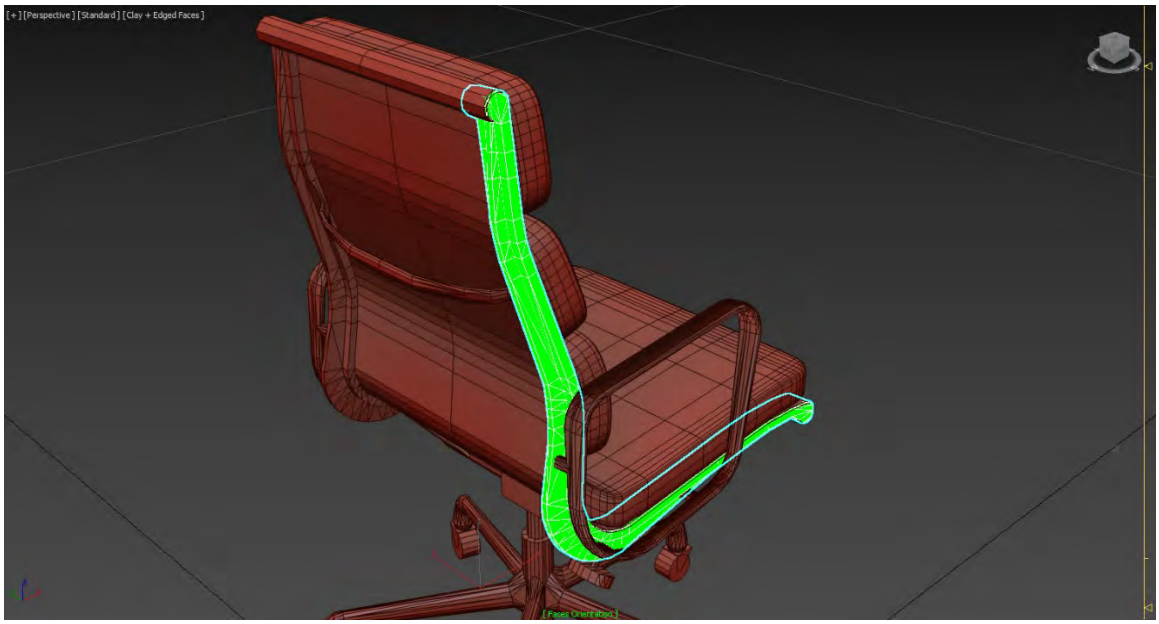
Stove looked complete, but Loop revealed that two vertices had not been wielded properly. Once these two vertices are combined, it issue is resolved.

X-View (Edit Poly)

3ds Max implemented a feature called X-View a few years ago. This was a tool designed to identify issues with 3d Models in the viewport that were using Edit Poly. One of the view modes in X-View is the ability to check for Gaps. Enable this in the Viewport Settings in the Top Left corner of the Viewport that is being used for modeling.



This XView is set to Open Edges. The green highlighted edges are not connected to any additional polygons, which means there is a gap here.



This XView is set to Reversed Faces. These are polygons that are facing the wrong direction. (Yes! There is a correct way that a polygon faces, and this is known as a Normal.) The green highlighted polygons need to be flipped. This can be done by selecting all the polygons and clicking "Flip" in the Edit Poly Tools.

NOTE: The mesh under review needs to be an Edit Poly Mesh. This can be done by converting the Mesh, or just adding an Edit Poly Modifier on top. The mesh then needs to be selected in the viewport for XView to show the issues on top of the mesh.

Overlapping Faces (Z-Fighting)

Another issue when importing meshes from another application (especially Revit in this case) is Overlapping Faces. This will cause visual issues in the 3ds Max Viewport, Rendering and in the Real-Time Engine. There will be strange looking artifacts on the 3d Mesh in the location of the 2+ faces directly on top of each other. This artifact effect is known as Z-Fighting, and is due to the Viewport/Renderer not being able to decide which polygon takes priority. So it essentially gets the two confused and tries to render both, but it can only render the data from one polygon. Hence the artifacts in the Viewport/Renderer.

The solution to this is relatively simple, just delete one of the polygons! Keep in mind that this might make a gap in the Mesh, and might need to be patched up after deleting one of the polygons.

It is easy to overlook Overlapping Faces, but identifying Z-Fighting is a straight forward process. There is a good chance you won't see it until it's too late. Sometimes Z-Fighting

will show up in the 3ds Max Viewport when it is rotated. But the easiest way to detect Z-Fighting is to just do a quick render. There will be extremely noticeable artifacts in the rendering that look like random noisy triangles or polygons.

The chair here has all the cushions duplicated on top of each other. Notice how the geometry doesn't look clean at all. This is what to expect when Z-Fighting is rendered. Deleting the duplicate faces will solve this. It is as simple as that, but still annoying.

Most instances of Z-Fighting from Revit will be:

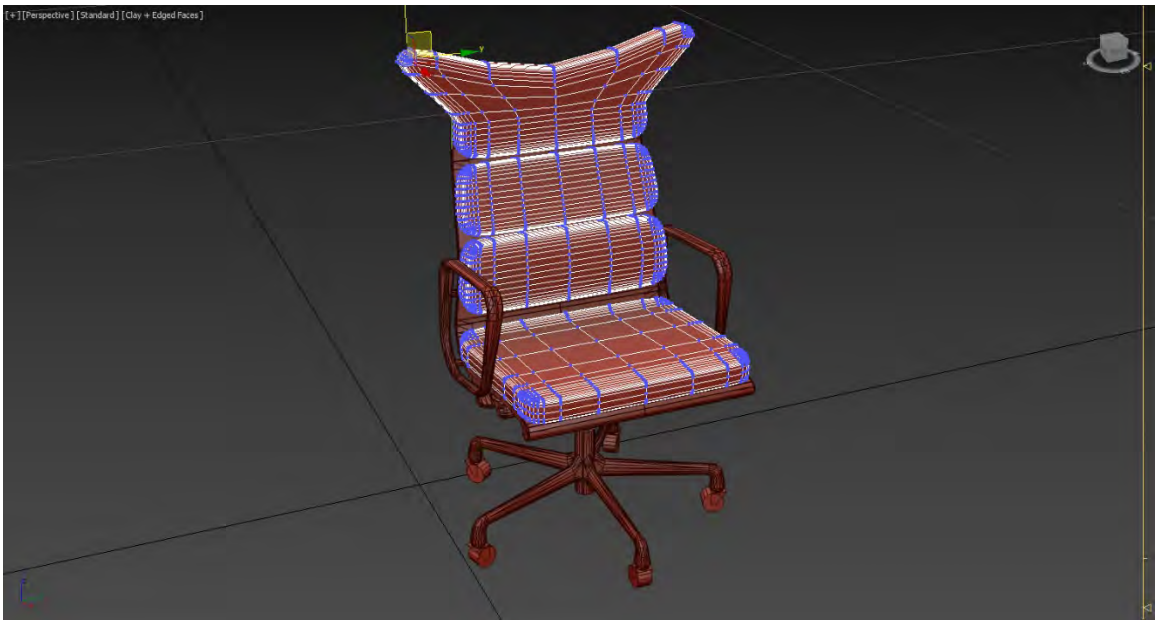
- Where two walls meet
- Where a ceiling meets a wall that is taller than where the ceiling stops
- Multiple Floors and Slabs from the Revit Model
- Where the edge of Windows, or Curtain Walls, but up against additional geometry. This could be a Wall or a Mullion.
- Additional Massing Models being included in the View that is being Exported/Imported

Symmetry Modifier

One of the best Modifiers in 3ds Max is the Symmetry Modifier. It will allow you only to model a part of a mesh, and mirror the results along an axis and position. This is INCREDIABLY helpful in 99% of all cases, and can save a considerable amount of time and energy. If the model is symmetrical in any way… USE THIS MODIFIER. Also, users can be creative with the modifier, and use MULTIPLE modifiers per object if needed. This isn't applicable in most instances, but for those few times where something can be mirrored on top of itself multiple times.

The chair's cushions have been sliced in half since we are cleaning up the mesh, and do not want to duplicate the effort on both sides.

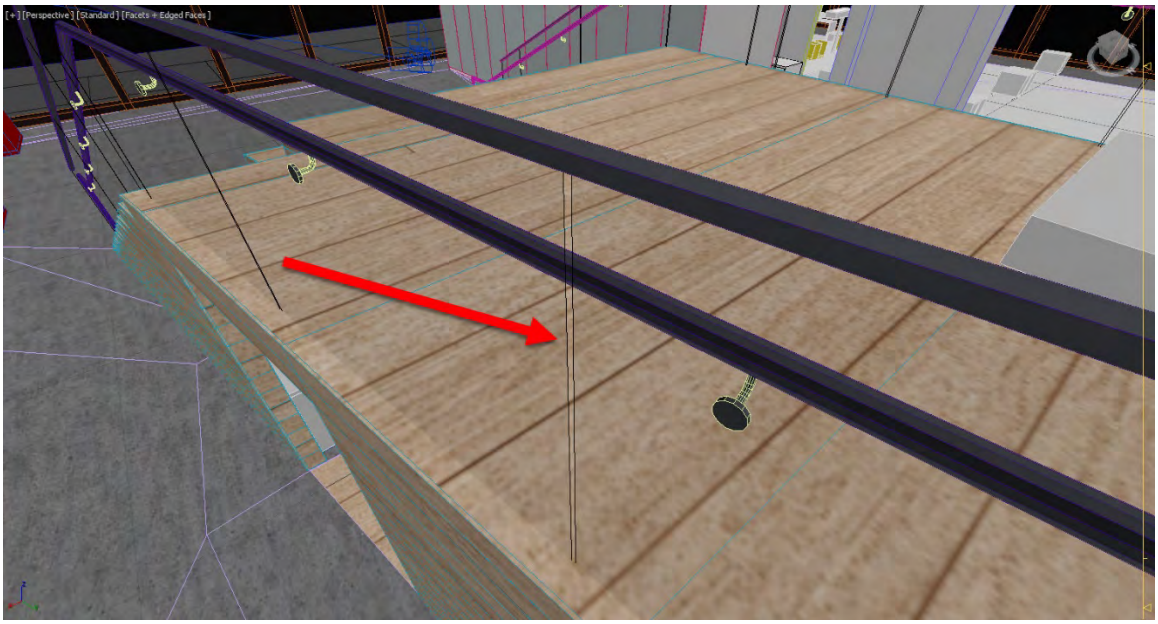
Here we see the Symmetry Modifier has been added, and the chair looks whole again!



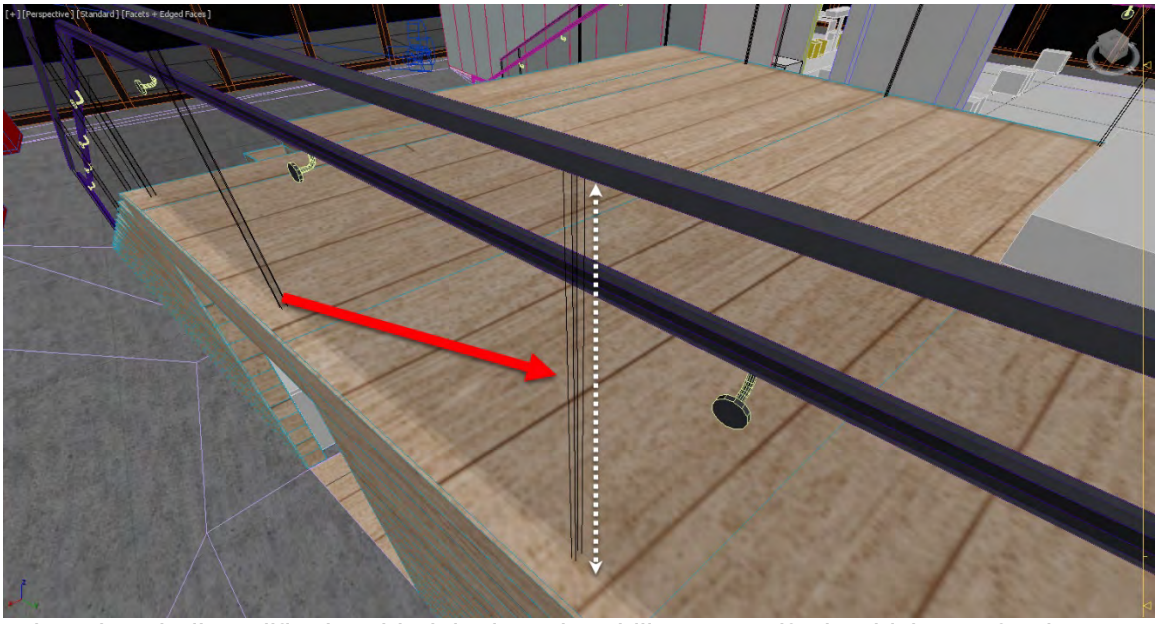
With the Symmetry Modifier added, any changes we make below it will be replicated on both side…And I mean ANY change…

Shell Modifier

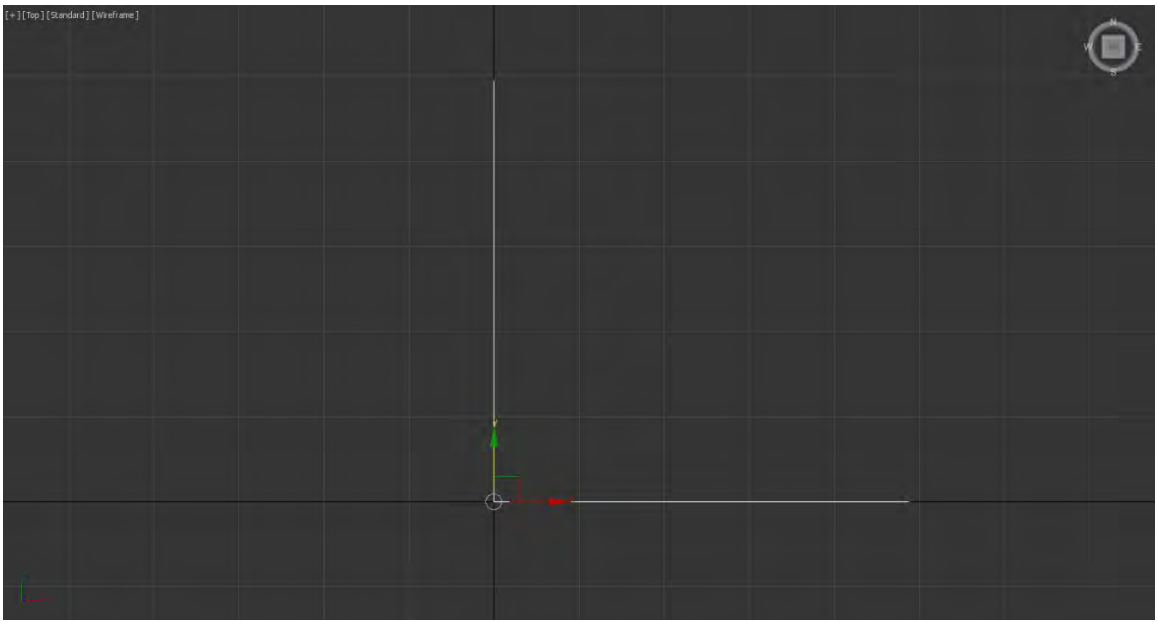
Another extremely powerful Modifier in the 3ds Max toolkit is the Shell Modifier. If there ever is a need to add a certain amount of thickness to anything, the Shell modifier has your back. It will give users the ability to extrude every polygon along its normal either positive or negative (Inside/Outside). The only issue arises when a polygon turns a sharp 90 degrees. The extruded edges will not be “straight” (See example below). This is easily fixed with an additional Edit Poly modifier, and aligning the new Vertices with the other two edges. (The snapping tools help a lot here!)



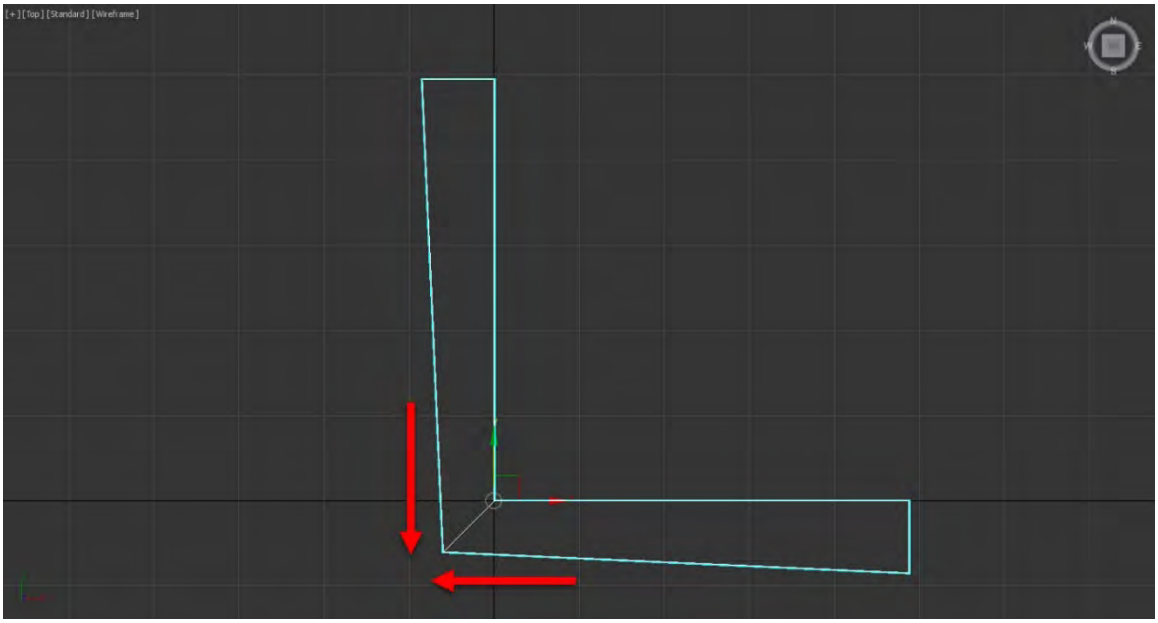
A perfect use for the Shell Modifier is for glass. This railing is a prime example. There is a single Polygon, which has no thickness. However, glass has thickness.



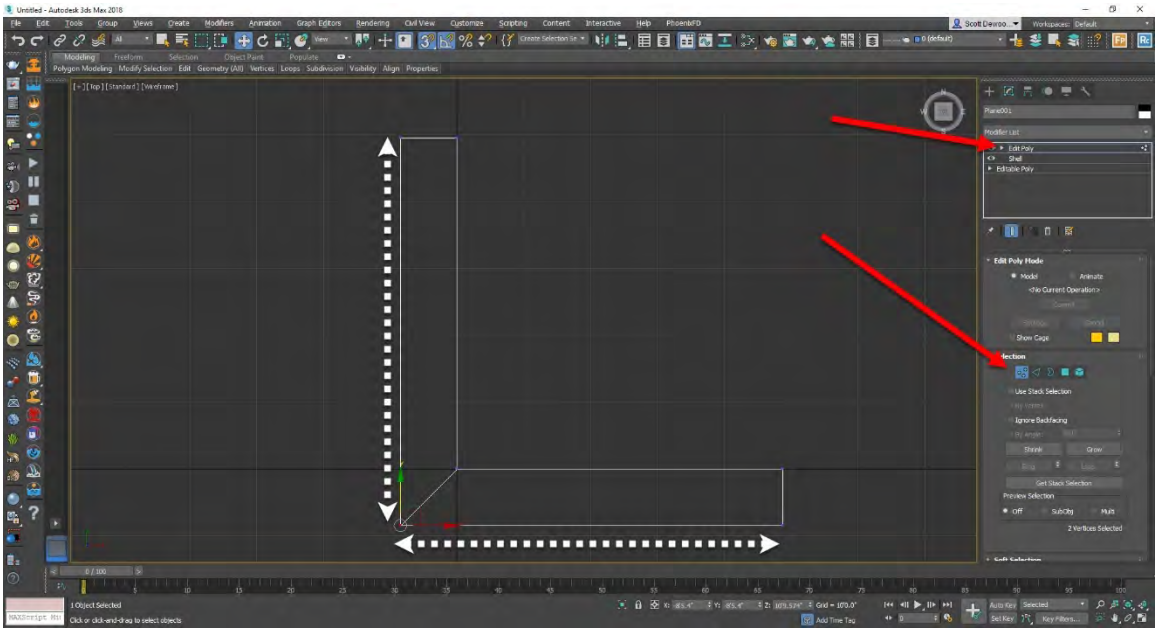
When the Shell Modifier is added, it gives the ability to specify the thickness for the glass! It can be specifically set to the dimension needed.



Looking directly down at a potential wall, the 90 degree turn will turn into a small issue when adding a Shell Modifier to extrude to make a wall.



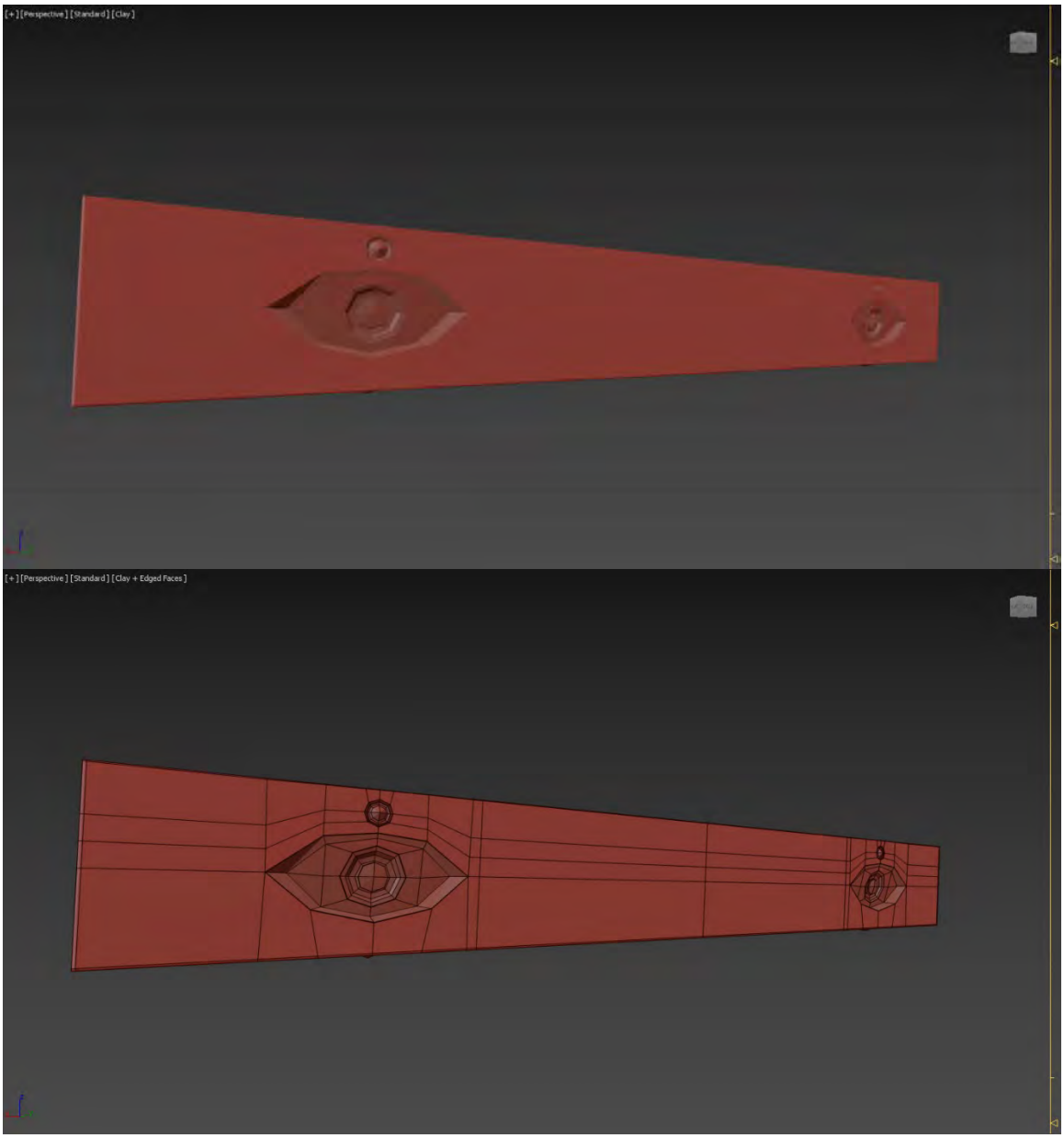
Notice how the extruded edges are not straight.



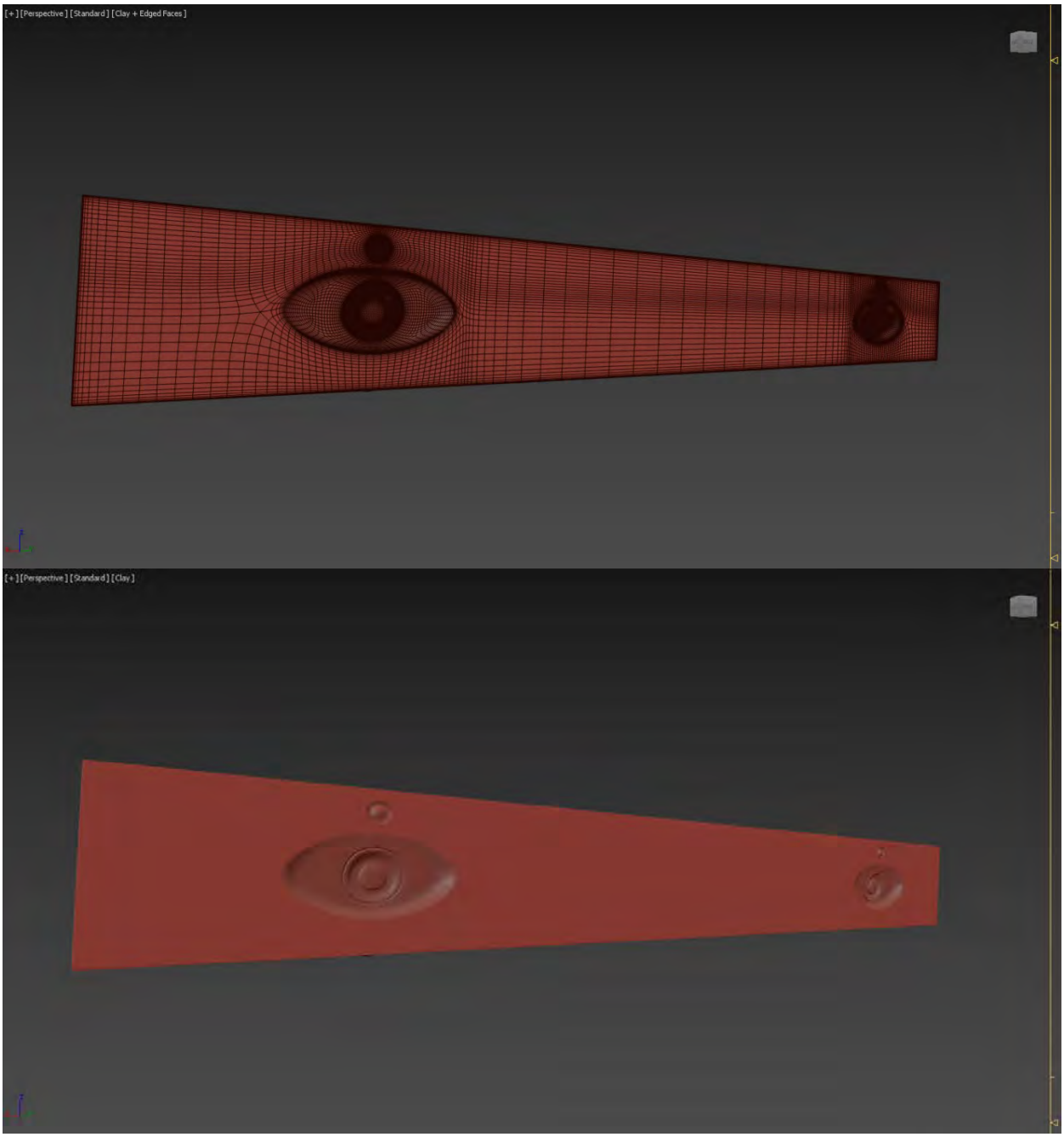
This can easily be fixed by added an Edit Poly on top of the Shell Modifier. Selecting Vertex Mode, move the extruded corner vertex to align with the other extruded vertices. It helps to use the Vertex Snap tool to align perfectly to the other vertices!

OpenSubDiv Modifier

It is one of the newest Modifiers recently added to 3ds Max over the past two years, and is probably one of the most helpful. Unlike its predecessors (MeshSmooth and TurboSmooth), the OpenSubDiv Modifier helps smooth objects without the need for a lot of additional polygons! This means base-meshes can have a lower poly-count, and still smooth as they might have in the past with Mesh/Turbo Smooth! And remember, low poly-counts are super important for us! The key benefit here is when creating both a Low and High Poly Mesh for baking Normal, Height, Cavity, and AO maps for the Real-Time Engine.



Here is a Sony Sound Bar that I've modeled in 3ds Max. Notice the rough curves and hard edges. Now let's add that OpenSubDiv Modifier…



Now look at the model! Edges are smoothed out and so are those curves! Though it is at the cost of additional polygons being added to the model. This is a good technique for creating the “High Poly” Mesh for Normal Map Baking, which we cover later in the document.

NOTE: How Smoothing Modifiers in 3ds Max work is they compare the distance between two edges, and interpolate a curve between them. The further away the edge are, the greater/relaxed the curve is. (Make a box primitive in 3ds Max, and just add a TurboSmooth Modifier to see what happens!) If the edges are close, the curve is tighter and more pronounced. So, it would be common practice to add additional edges around other edges that needed to have a tightly curved edge. (Illustrated Below). The OpenSubDiv replaces this workflow some by assigning a Weight value to the desired edge. This Weight Value controls the interpolation effect of the curve for that edge. (Illustrated Below)

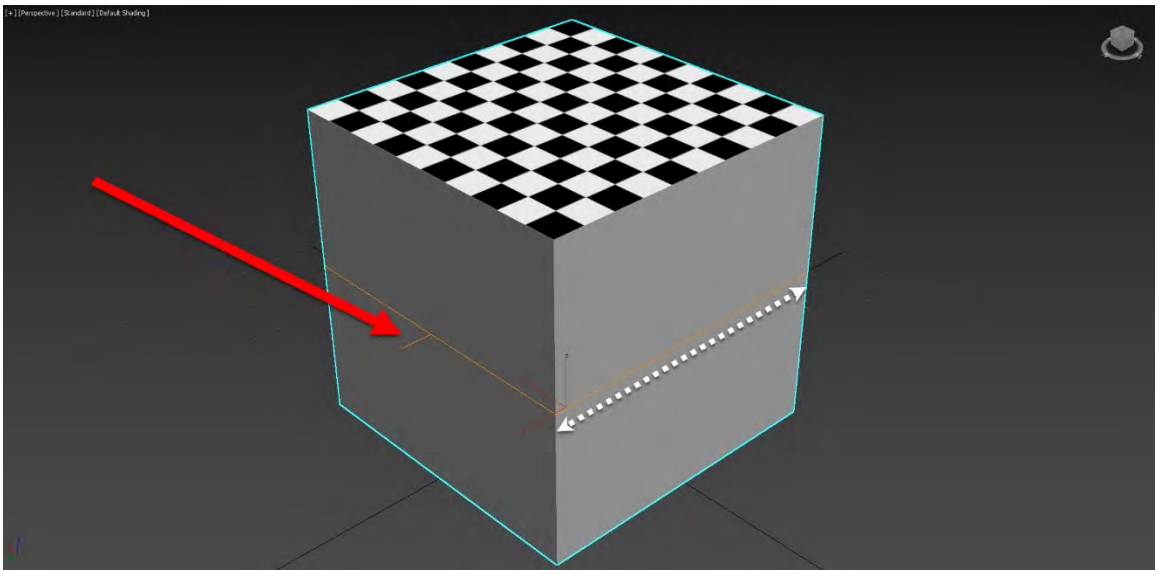
UV Mapping / Unwrapping

NOTE: Whole books, and hours of online training, have been created around this one topic alone. I'm not going to be able to cover everything under the sun that could be cover in this section. There could even be a whole dedicated AU Class for this section alone, which still probably wouldn't be enough. I am going to try to give enough of an over-view to point you down the correct path. There is a good level of 3ds Max, and UV Mapping, understanding expected in this section. If you are brand new to 3ds Max and UV Mapping, trust me when I say a simple Google (or Bing) search will yield and incredible amount of resources around this topic. So please use this as a starting guide to get you going!

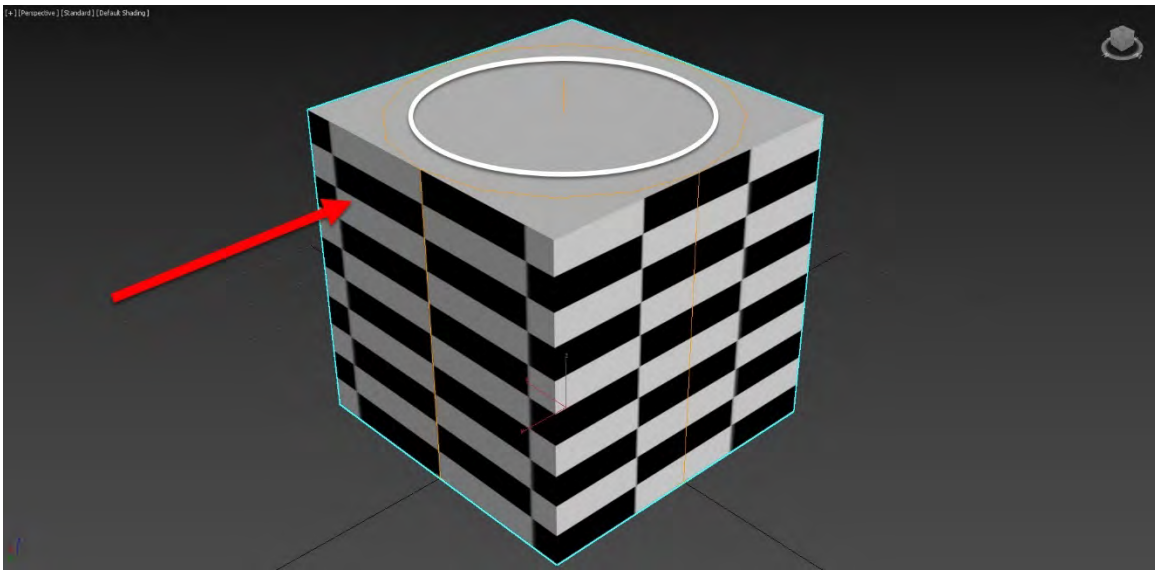
UV Mapping is a process in which a 2D Texture is projected onto a 3D Surface. If you have been working with Revit, Rhino, or Sketch-Up, you have been UVMapping every time you have adjusted the Scale and Rotation of a Texture. There isn't much more that can be done with the UVs in these applications, but in 3ds Max there are way more options available.

UVMap Modifier

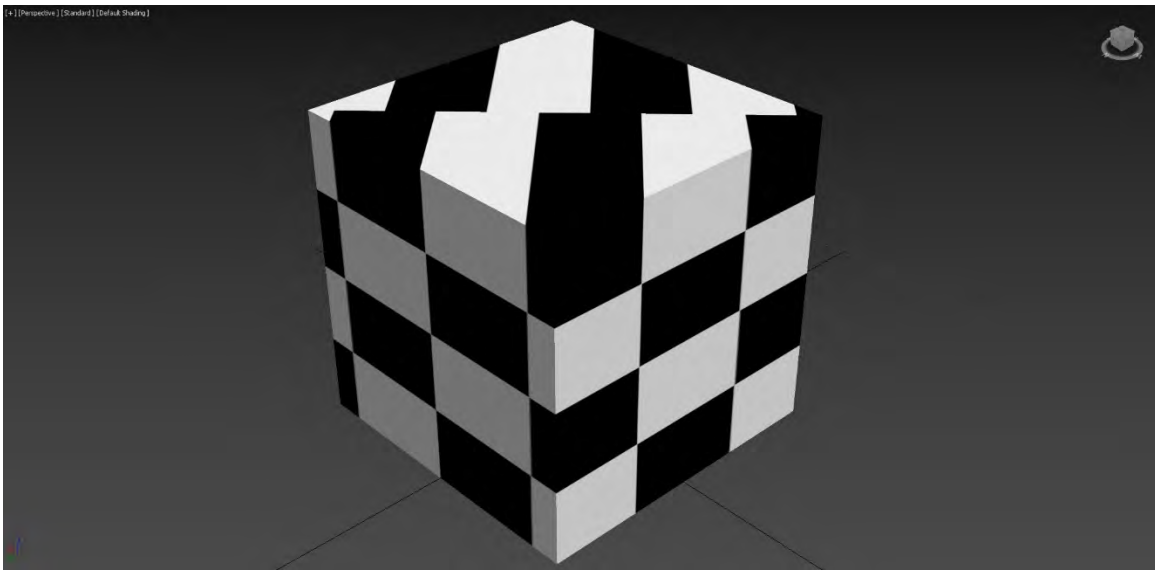
First, in 3ds Max there is a UVMapping Modifier that can to be applied to the 3D Object. This is the most basic way to apply UVMapping to an object, and the modifier has a few preset projection methods: Planar, Box, Cylinder, Spherical, and by Face(Polygon). For most objects, this might provide enough mapping that the model does not need to be "Unwrapped". But if we get to more complex objects, such as Furniture, Characters, or anything that needs textures placed in a specific place or direction, the UnwrapUV Modifier needs to be applied. Or even more importantly, UV Unwrapping needs to happen when preparing models for Baking. (We'll cover this in the texture section.)



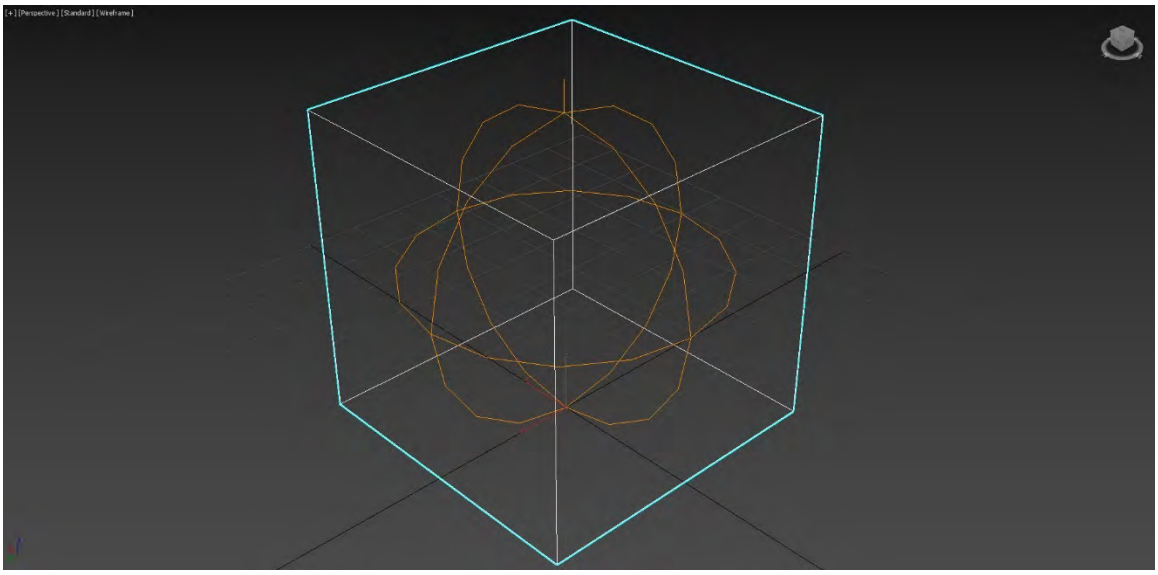
This is what UVMap Modifier will look like when first added to an object. It starts off in "Plane" mode. Which is just a flat 2D projection for a texture on an object. Notice the Orange Outline that is on the object. This is the size/scale of the projection. Also take notice of the little line that extrudes form one edge. This is the "top" of the texture.



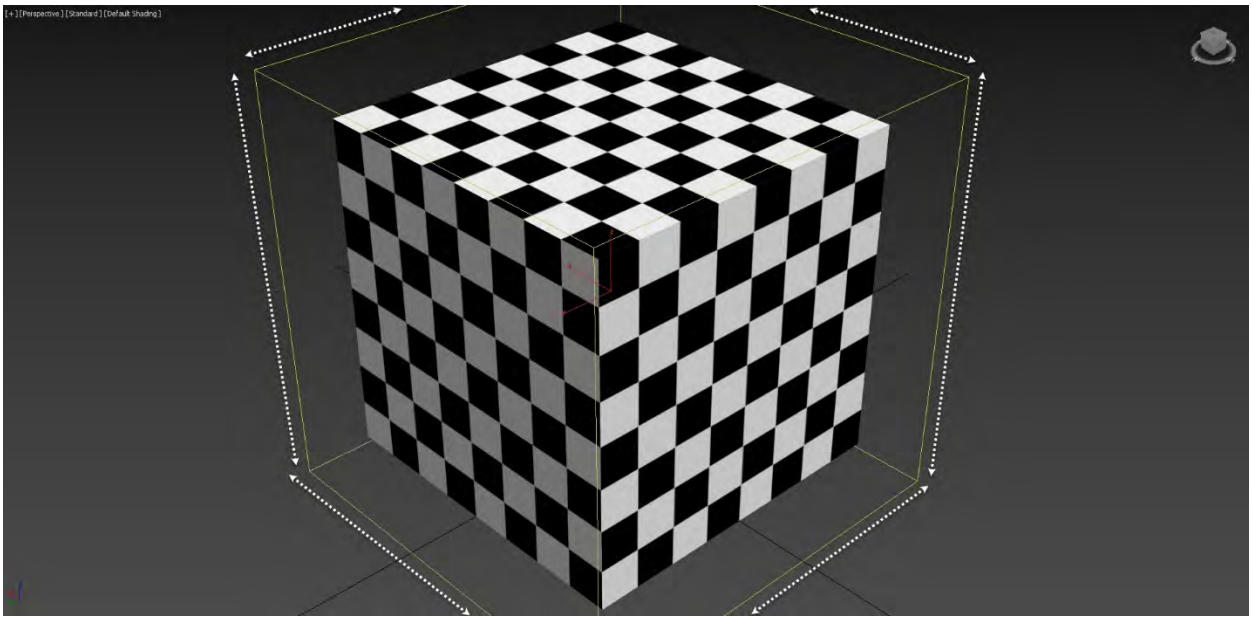
Here is a Cylinder projection with the UVMap Modifier. It might be hard to see the orange outline on this screenshot, so I have highlighted it for you. Notice the top has no projection (which can be enabled in the modifier), and then how distorted the checkers are on the sides. This is due to how round a cylinder is, and how it conforms to a box.



This is a Spherical projection. It's hard to see though until we turn on wireframe…



There it is!



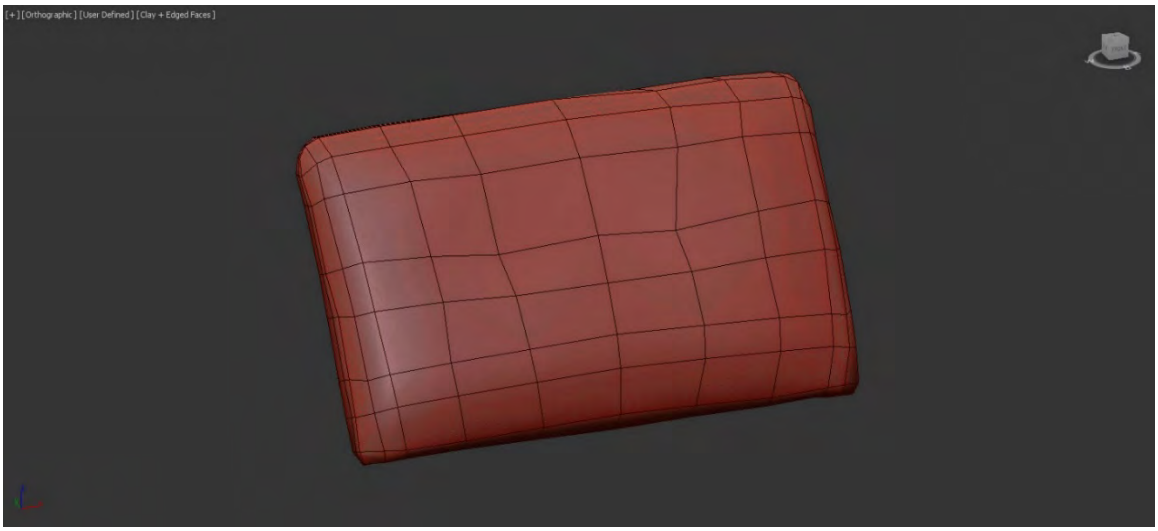
The most commonly used (at least by me) for the UVMap Modifier is the Box projection. This will project the texture on the XYZ axis, which will surround an object. Just keep in mind that there will be seams where the projections meet.

UnwrapUVW Modifier

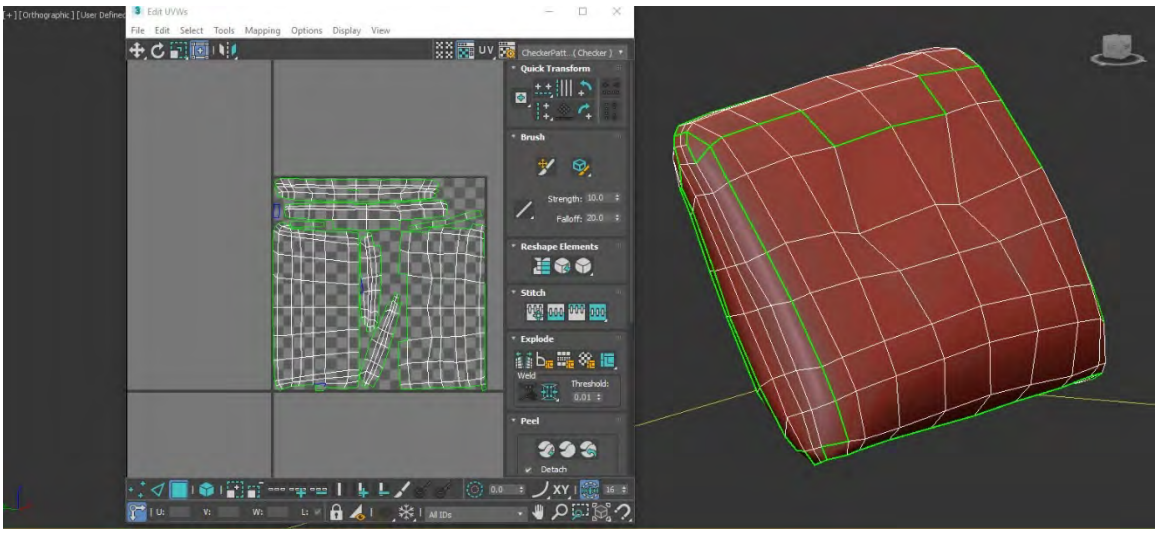
Unwrapping a model sounds exactly like it sounds. It is the process of taking all the polygons from the 3D model, and laying them out in a 2D Layout, which is set as a square. The U and V represent X and Y in a graph, and is set from 0-1 in each direction. And this is where good, clean, topology will make life 100x easier. Because when you are unwrapping a mesh, each polygon needs to find a spot in the UV Grid. Thankfully 3ds Max has a nice set of tools to help automate some of this work. Most likely some vertices and edges will need to be re-aligned and wields to remove Seams.

Seams are going to be breaks in the UV Map where there will be a clear distinction that the texture map does not flow properly across the mesh. If possible, try to place the seam in locations where someone will never see it. On hard surface objects this might be a bit easier than something along the lines of a Character. But even for Characters, there can be some obvious places to put a Seam. For instance, putting a UV Seam along where an actual seam in the Character's clothes would be located is a good choice! Seams are unavoidable in Unwrapping, so keep them in mind when modeling your objects.

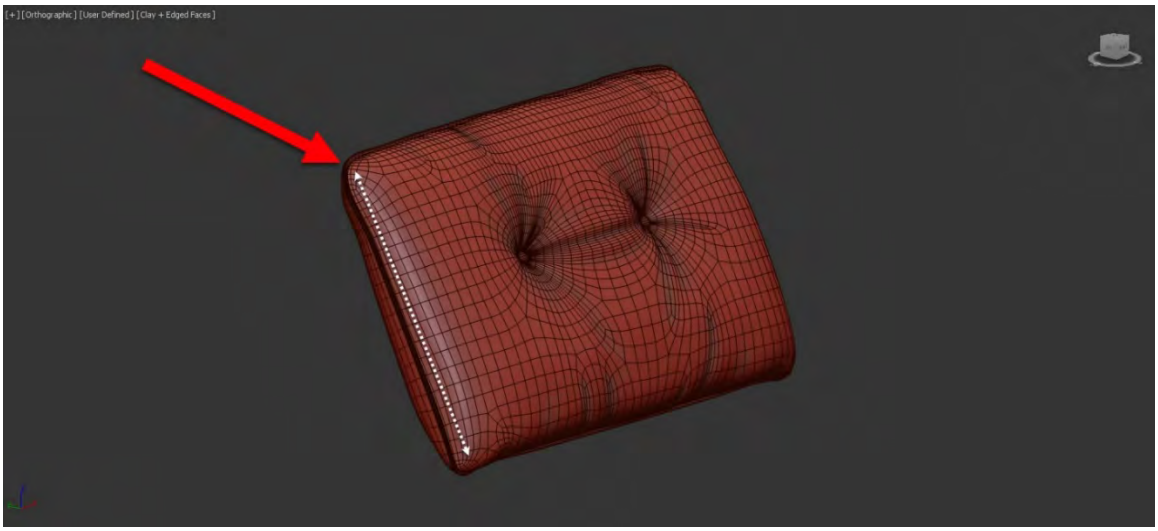
> **NOTE:** There are some technologies, such as UDIM, that allow users to move out of this space. But those are a little more advanced for this class, and are not entirely supported by Real-Time Engines.



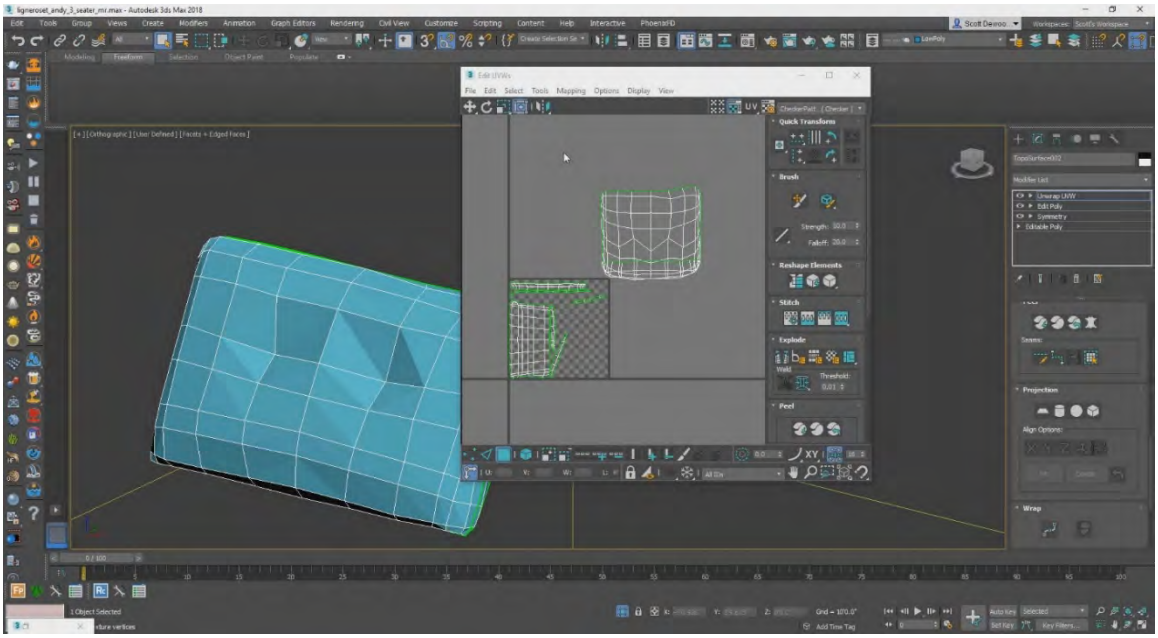
Here is a Pillow that we have Re-Topologized. We're going to want to UVUnwrap this pillow for texture baking. So, let's add the modifier and see what we get…



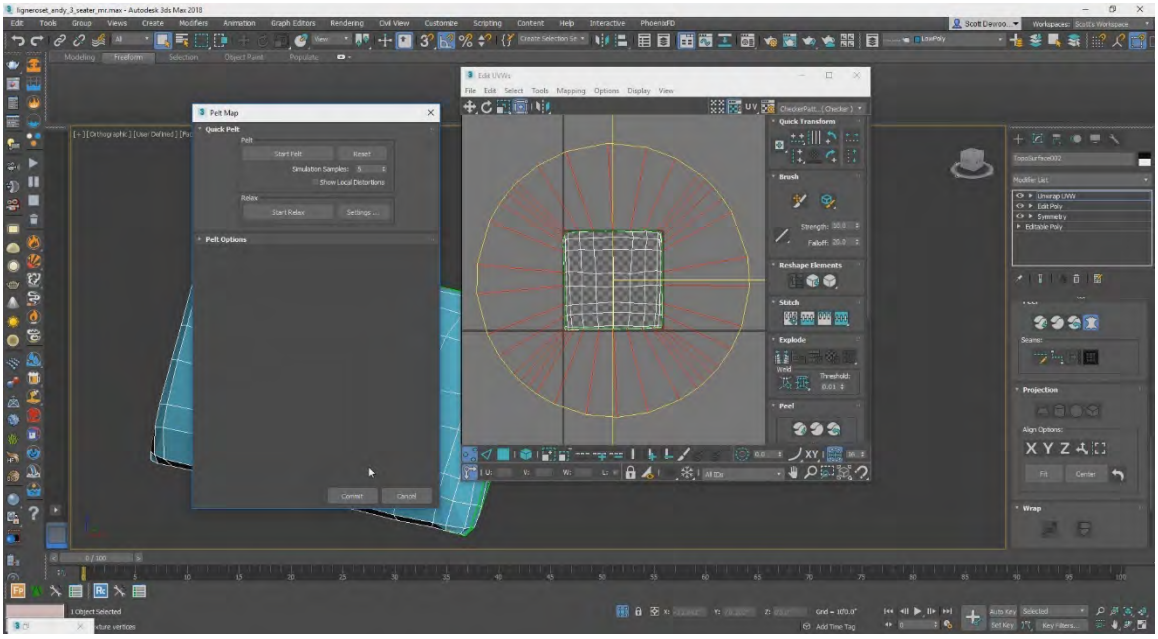
Well, this isn't too pretty… Those neon green lines are seams, and there are a lot of them! They aren't really placed well, and cut through some key parts of the mesh. We'll need to look at cleaning that up. So where might be the best place for a seam? Well, let's take a look at the High Poly Mesh of this pillow…



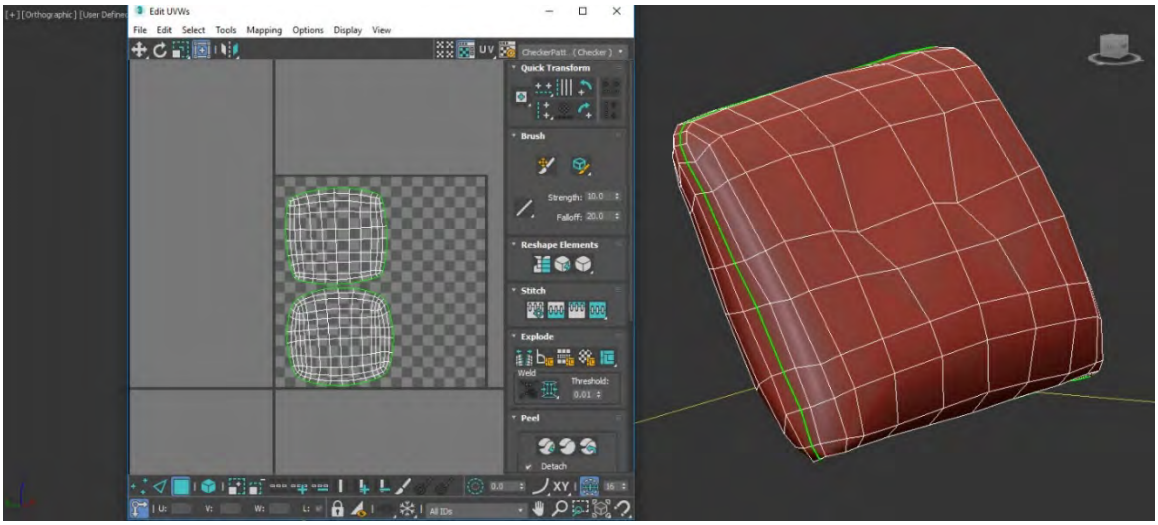
Looks like the piping that goes around the entire pillow will be the perfect place for this! For two reasons: 1) It's where the seam is in reality 2) It can easily be hidden here because of reason 1.



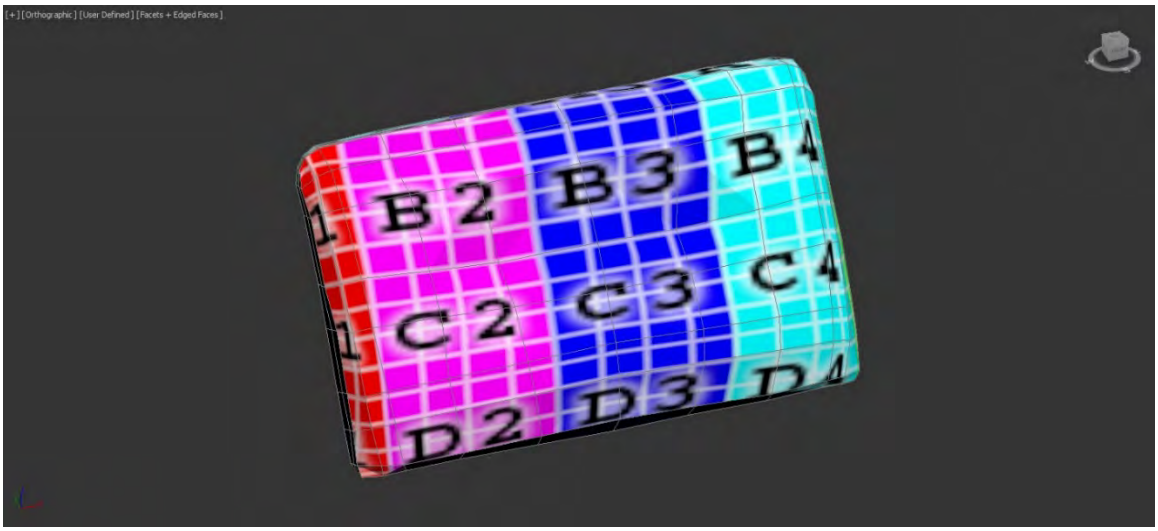
So, we need to separate the two sides of the pillow. This can be done by selecting all the polygons and then applying a Planar projection, which is in the Modify Tab in 3ds Max. But this isn't the result we want. There are polygons overlapping each other… But there is something called Pelt Mapping…



That did the trick! Pelt Mapping is taken from the same idea of actually creating pelts from animals. It will add in points around the UVs and then stretch them out evenly. This is a perfect use-case, but others are for Characters and other “organic” type objects.



After Pelt Mapping both the front and the back of the Pillow, we need to align the polygons inside of the UV Grid. Make sure to keep them in the 1x1 Grid, and that they do not move outside of this space. Let’s apply the Texture Checker to the mesh, which is inside of the UV Editor…



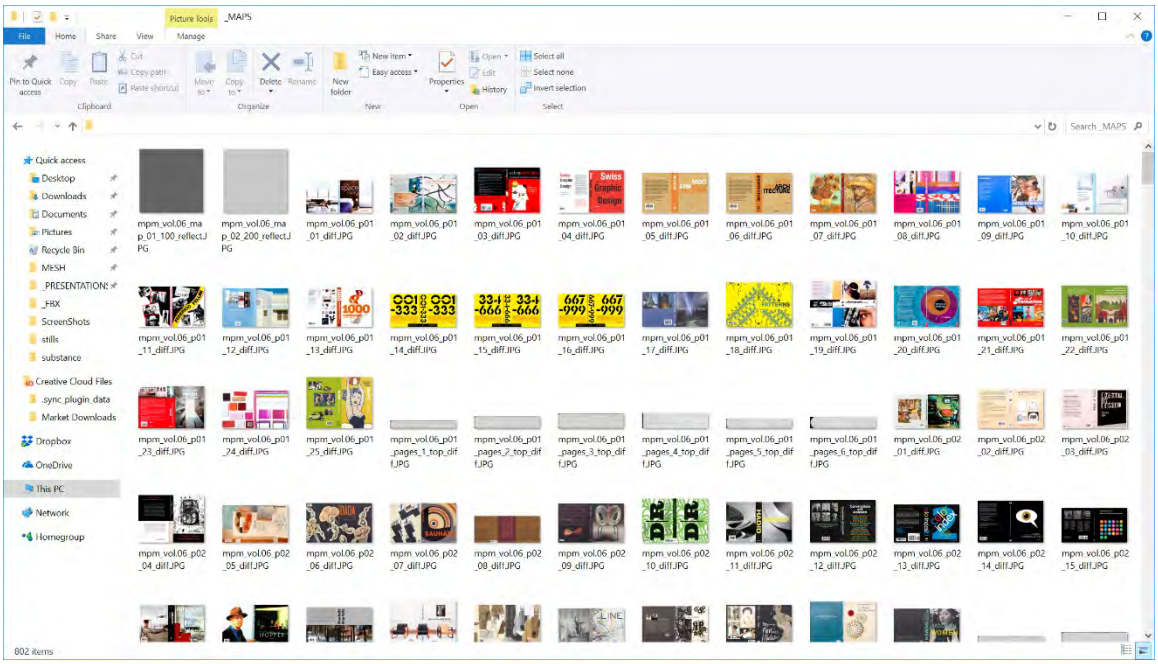
Now we can see how a texture would deform over the mesh. It's not 100% perfect, but that's ok in this instance. We'll most likely apply a leather texture, which is pretty organic in nature. So, it should be just fine. This will not affect baking the Normals either, which we will cover in a later section. We'll see this pillow again soon!

Atlas Maps

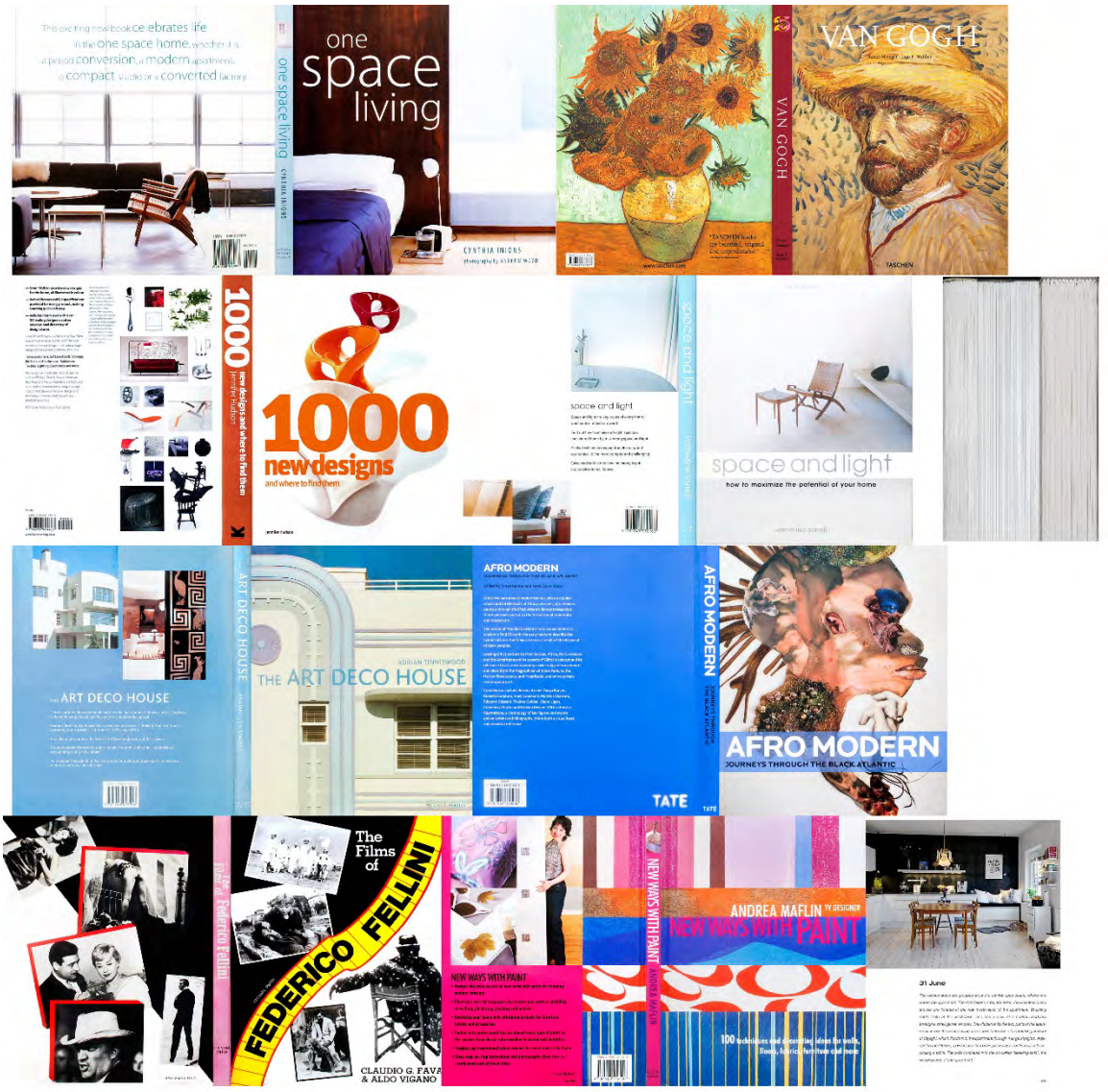
One of the important things to remember when Unwrapping 3d Objects is that you can unwrap multiple objects at the same time, or have objects "share" UV Space. When adding multiple objects into the same UV Space, a single texture map can be applied to all the objects, reducing memory and rendering time for the GPU in the Real-Time Engine. (This can also be helpful for Offline Renders, such as V-Ray!) This is known as an Atlas Map, and it is super important when optimizing your VR Experiences Performance to consider doing this for as many objects as you can get away with. Good use cases of Atlas Mapping would be for books, DVDs, and other small objects. Entire rooms, or pieces of furniture, could also be combined into an Atlas Map. These will dependent on the scene you are creating, and the objects with in it. Just make sure that everything stays within the UV Grid!

> **NOTE:** To help with the creation of Atlas Maps in 3ds Max, a UnwrapUVW Modifier can be applied to multiple objects at once. The objects that are currently selected will show up in the UVGrid.

> **NOTE:** There is an Atlas Map tool on the Substance Share website for free! It's made by Wes McDermott and can be found here.



Here are a bunch of book textures from Model+Model's Book Collection. Notice that each book cover is in their own individual image. This also goes further inside of 3ds Max, where each book has their own Shader as well. We can take these images and start to combine them into one image… aka an Atlas Map…



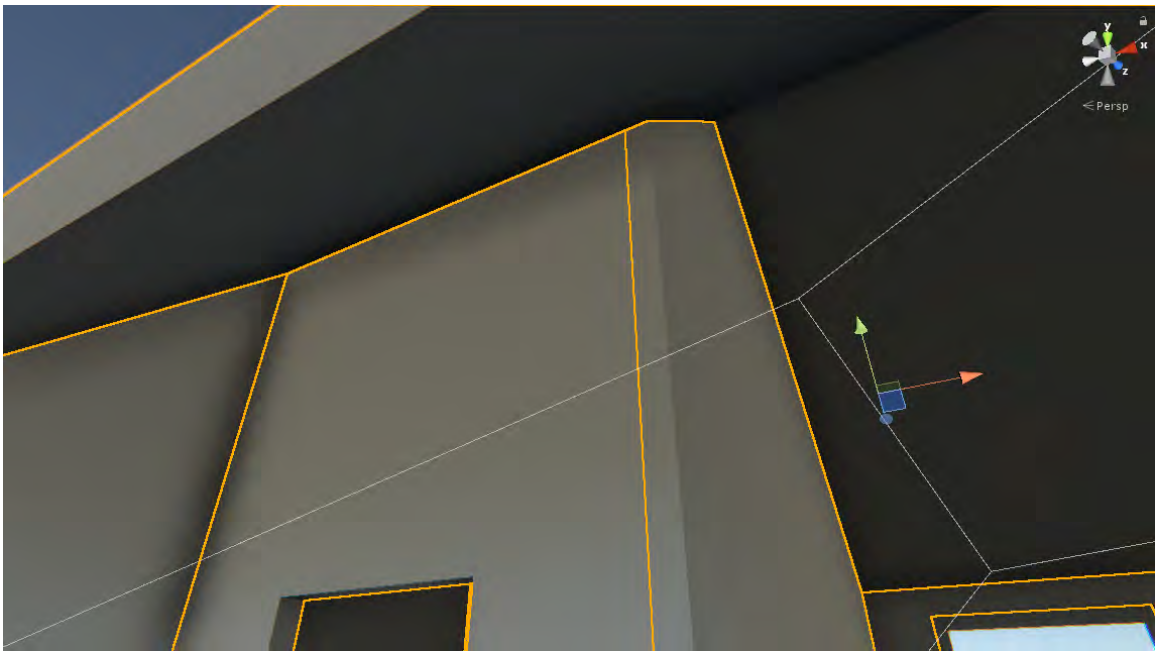
Once the Atlas Map is created, the UVs for the books just need to be aligned over which cover needs to be used. Also included in this Atlas Map are the book pages, which can also share the same UV Space! Consolidation such as this will lead to much better performance in Real-Time! (And even in Ray-Trace Engines!)

Lightmaps

In the Real-Time Engines, part of the process will be to bake the Lighting in the project files onto the Static Meshes to increase performance, while retaining the amazing lighting quality they can achieve. But for this to happen, the mesh needs to have a special set of UVMapping done to it. This can be handled in one of two ways. The first way is the Real-Time Engines can generate the Lightmap UVs automatically on import for all the 3d Meshes. In most cases, this will be the ideal solution to handle this. But there will be the instances where 3D Meshes might require specific placement of the UVs for the Lightmaps. (Characters are a good example of this.) This can easily be done in 3ds Max by adding an UnwrapUVW Modifier and setting the UVMap Channel to 2. The Real-Time Engines will recognize multiple UV Channels on a single mesh, and will use UVMap Channel 2 as the channel for the Lightmap. (UV Channel 1 is always used for the primary mapping of an object.)

> **NOTE:** Lightmap UVs cannot have ANY polygons overlapping each other. This is because each polygon will be receiving its own lighting information, and if two polygons share the exact same spot, the lighting will look incorrect in at least one area.

Earlier in the Revit section, I mentioned what happens when Revit Walls come together in 3D. This is the result of the issue that is eventually caused by this inside of a Real-Time Engine. (Unity in this example.) It is subtile here, but there is a thin line where the red arrow is pointing. And it runs down the entire length of the wall. This is the shadow line that gets generated due to the Lightmap UVs not coming together.



This again re-illustrates where the walls are coming together with the highlighted yellow lines. The area we're focusing on should actually be at the corner, and not set back a few inches.

Animation

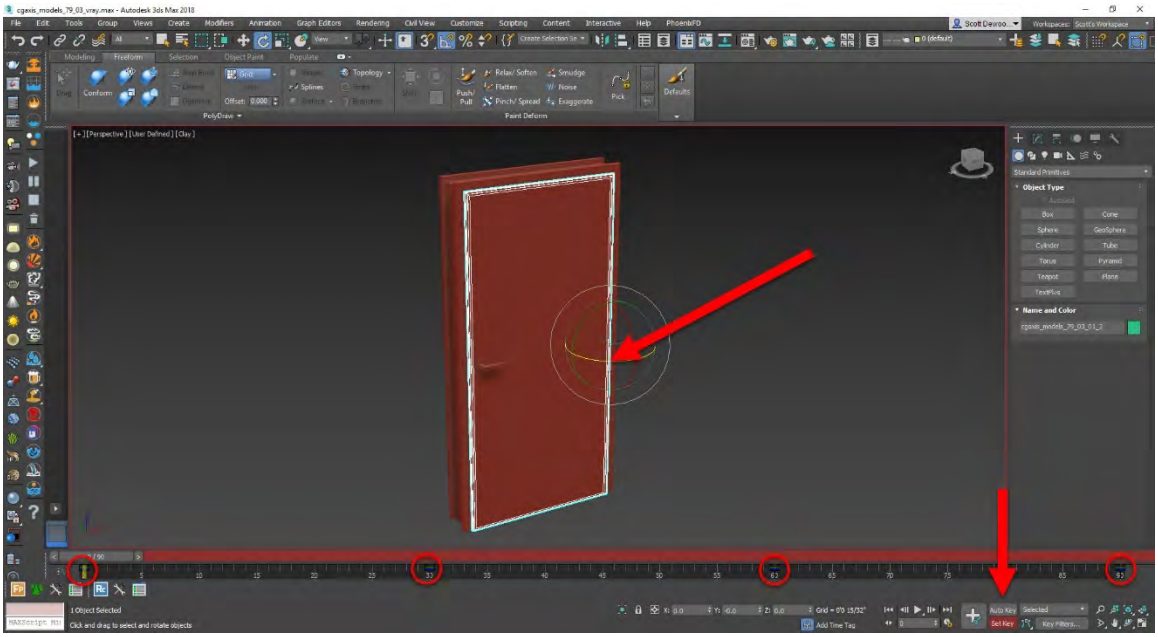
NOTE: Whole books, and hours of online training, have been created around this one topic alone. I'm not going to be able to cover everything under the sun that could be cover in this section. There could even be a whole dedicated AU Class for this section alone, which still probably wouldn't be enough. I am going to try to give enough of an over-view to point you down the correct path. There is a good level of 3ds Max, and Animation, understanding expected in this section. If you are brand new to 3ds Max and Animation, trust me when I say a simple Google (or Bing) search will yield and incredible amount of resources around this topic. So please use this as a starting guide to get you going! (Starting to see a pattern here yet? ;))

Some 3d Meshes may need to include Animation to enhance the experience when in a Real-Time Engine. As just about everything else in 3ds Max, it has an extensive set of tools for Animation. But unlike traditional animation, you will want to view animations as modular pieces. (Just like the 3d Assets themselves) Animations will need to be repeatable to properly loop repeatedly. Because when you're in a Real-Time Engine, there is no set start, or stop, as there is with a movie. A user could stand still and watch an animation on an object for as long as they would like.

Set/Auto Key

This will be the primary tool for animation. When creating an animation, the Set/Auto Key will place a keyframe for the animation being created on the timeline. Set Key is more of a manual input for the keyframe, where Auto Key will make a key frame every time you make a change to a property. One isn't better over the other, and most likely you will jump between the two of them. Thankfully their buttons are right next to each other for this reason!

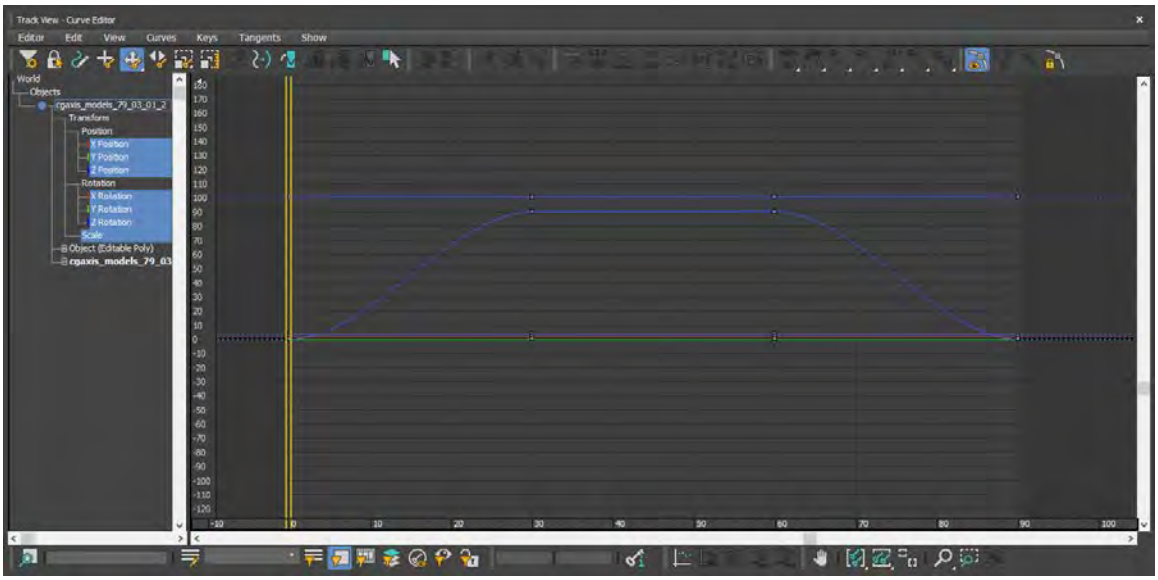
NOTE: The biggest thing to note here is to pay close attention where on the timeline you're placing the keys and are working! If you're edit a property, and are in the wrong frame, you will lose the edit once you shift the slider to the correct frame!



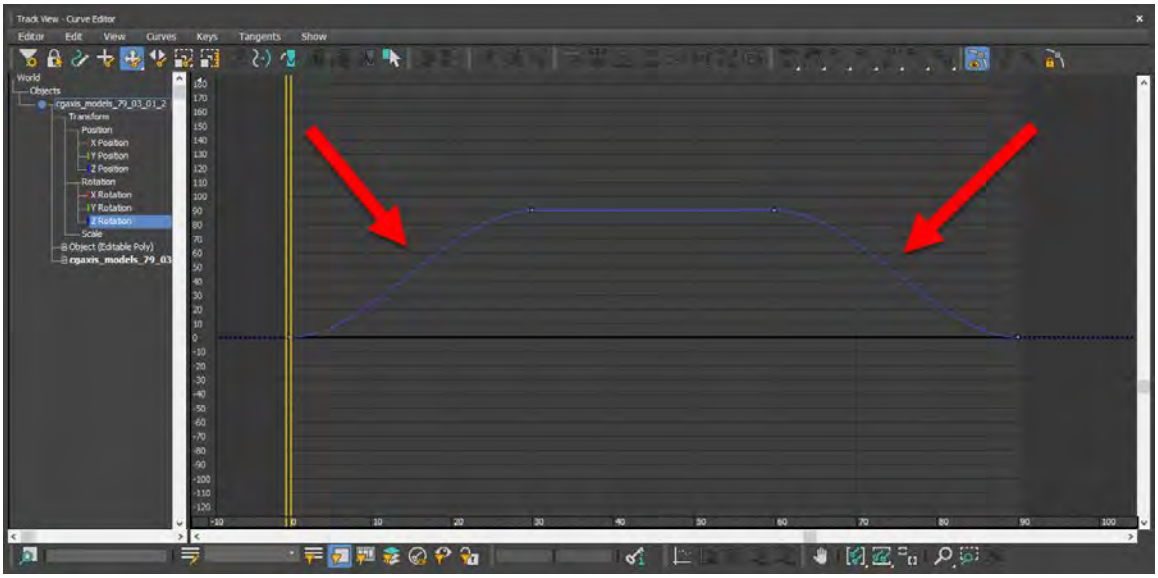
Here the door has been animated to open and close. The key frames are set 30 frames a part to easily identify the different animation segments. (The Game Exporter Tool can help with this!) This was done using the Set Key Tool, which is highlighted Red when enabled… Literally. Also noticed the Red Boarder around the viewport. This signifies we are in Animation Mode in 3ds Max.

Curve Editor

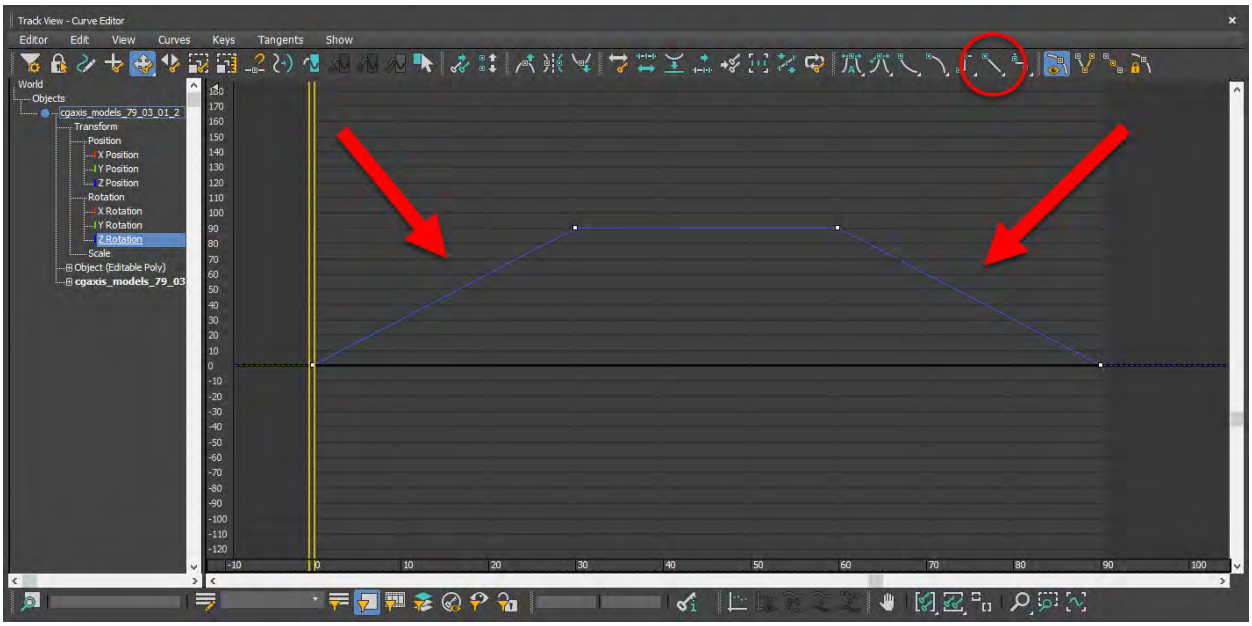
A common, and extremely powerful, tool in animation platforms is a Curve Editor. This tool is going to be what helps smooth out animations, and allows for fine tweaks in the animation. It's a little daunting at first, because it is translating X, Y, Z, and properties into 2D space across a graph with a curve. That just sounds complex! Working with the Curve Editor does take some time to wrap your head around, but the pay-off is huge once it becomes natural to work with. If you are unfamiliar with the Curve Editor, start small with simple animations. Opening and closing a Door is a good example of where to start with the Editor. Animating Characters takes the Curve Editor to a whole new level…



This is the Curve Editor. If this looks a little complex, just wait until there is a real animation loaded in this. This is the curve from the door in the previous example.



If we limit the curve to just the Z-Axis on Rotation, which is what we animated, we can see it a little more clearly. Notice the slope on the curve. This is known as an "ease-in" and "ease-out". It essentially has the door open slowly, hit a certain velocity, and the slowly end up fully open. If we want the door to open at a constant speed across the entire segment, we can make these Linear…



Now notice how the slopes are no longer there, and it is now just a straight line between the points. This means the animation is set to a constant speed, and there will be now ease-in or ease-out. This can be set this way by selecting the keyframes and then hitting the linear button at the top. There are other ways to interpolate the keyframes as well, which are next to the linear button at the top.

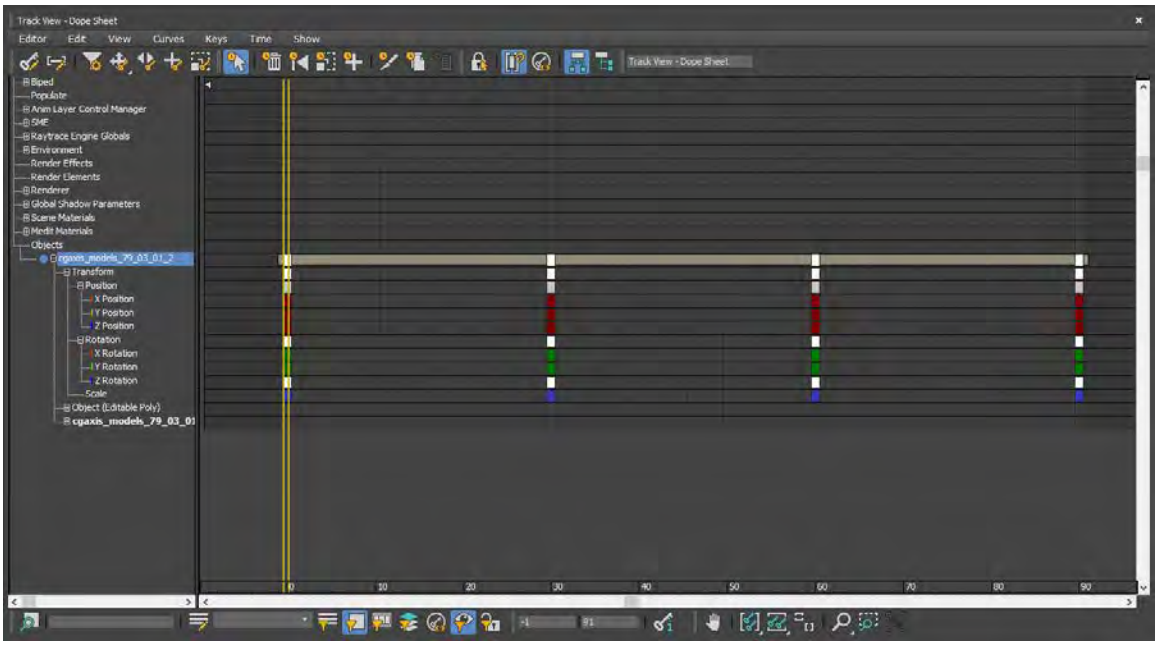
Dope Sheet

The primary focus for the Dope Sheet is to help arrange keyframes of properties along the timeline. Where the Curve Editor is more focused on the translation and interpolation of animation between the frames, the Dope Sheet is more focused on Timing. And with Animation, Timing is everything. How fast, or slow, something happens in VR is greatly going to effect the VR Experience. I only have one rule of thumb for Timing in VR:

Things will probably need to move slower than you think they are inside of 3ds Max.

So, if you're animating a door, and you think it looks good, slow it down some. This might not apply to everyone, as our sense of timing is off compared to each other. So, I do recommend doing a lot of testing in VR with your animations prior to letting the final product go out the door. Also test with multiple people to see if there is a good average in the Timing that can be meet.

NOTE: This is a rule I apply to just about everything in Rendering: If you think you've done something a little too much, you probably have. So just step it back a little bit, and you'll most likely hit the proper amount right on the head!



The Dope Sheet is straight forward. The keys can be broken down into their individual animation types and per axis!

Exporting

How do we get out of 3ds Max? It's a pretty easy answer actually: FBX. All the Real-Time Engines accept this format, and it will be the best format to use to send Model, Texture, and Animation Data over to the Real-Time Engine.

There are sadly a few things that will not translate well right out of the box:

Materials

This is the biggest one, especially if the models come from Revit. By default, from Revit, the Material applied in 3ds Max is not going to be supported by the Real-Time Engines and will need to be converted. Each Real-Time Engine is going to handle this a little differently, but the safest option is to use the good ol' Standard Material 3ds Max. This will be packaged up properly in the FBX, and at least the Diffuse Map will translate in accordingly.

I'm personally not too big of a fan of this, as I'm going to be replacing all the materials once I'm inside of the Real-Time Engines with Substance Materials. So, I normally have my materials as V-Ray Materials. These Materials won't directly translate through FBX, but their NAME will show up properly once in the Real-Time Engine. This is yet another reason why having good naming conventions for everything is important!

If you're going to Stingray (3ds Max Interactive), 3ds Max has the Stingray shaders built inside of it now. Users can use these shaders for a 1:1 transfer between 3ds Max and Stingray.

Cameras

These will not directly export 1:1 into a Real-Time Engine, but there are some known work arounds for this. A Dummy Object can be parented to a camera, which will then be placed as an empty game object in the Real-Time Engine. From there, the Real-Time Engine cameras can be aligned to the Dummy Object for the 3ds Max Camera's placement. It is a little backwards sounding. So, I wouldn't worry about setting up Camera Views in 3ds Max, unless you ultimately must use existing cameras from 3ds Max.

NOTE: Unity 2017.2 now supports camera importing from FBX!

Lights

This is something that will not translate from the FBX into the Real-Time Engine, and will need to be re-created once inside the Real-Time Engine. The same trick can be used for the Cameras for placement at least. But each Real-Time Engine is going to handle lights in their own way.

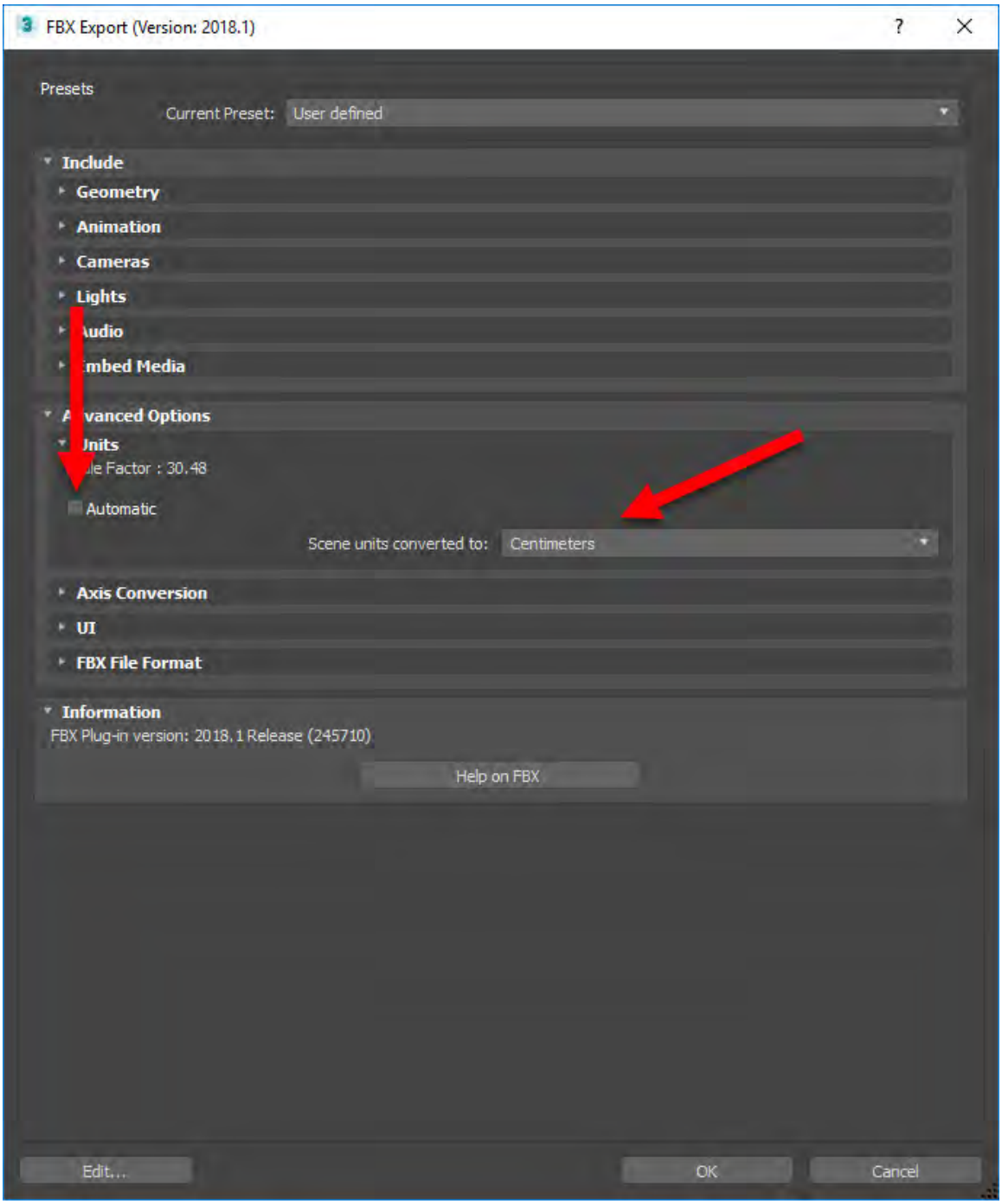
The most important thing to keep in mind when exporting to a Real-Time Engine, is the Unit Scale in which the Real-Time Engines work in. Thankfully when exporting FBX, 3ds Max will give users the option to change the Unit Scale at the time of Export. So, Users can work in Feet, Inches, 3d Max Unit (Please don't do this), or Meters inside of 3ds Max. It will be super important to try and keep the Unit Scale the same though the pipeline as much as possible. But when it's time to get to the Real-Time Engine, there is one final Unit Scale everything must be set to:

Unity = cm

Unreal = cm

Stingray = m

Make sure that the Unit Scale in the FBX Export matches the targeted Real-Time Engine listed above. If I have not listed the Real-Time Engine that is being used, this Unit Scale could be different. Please refer to the Real-Time Engine's documentation for details on what their System Units might be.



Make sure that the File Units when Exporting FBX look like this when going to Unity or Unreal. This should say Meters if moving to Stingray.

NOTE: There are some other workflows, such as the link to 3ds Max Interactive (Stingray) and Epic Game's Datasmithing toolset. However, we're not going to dive too much into those, as I'm taking a more generic mindset in terms of where these models might end up. FBX gives the most flexibility here. But please do not rule out these other options, especially if they fit your project's pipeline well!

PBR Textures and Model Baking

Materials in VR will go a long way into the Visual Experience of the project being created, and it will also have a major impact on the performance of the VR Experience as well. Real-Time Engines have always handled Materials in their own way. Thankfully over the past few years now, a new industry standard has been adopted for Materials called PBR (Physically Based Rendering). This new standard takes a very uniform approach to Materials, as it grounds all its properties from real-world physics. Hence the name Physically Based Rendering. There are a lot of documentation around how these properties work in CG, and in the Real-World, all over the internet now. I did a webinar earlier this year for NVIDIA on this very subject, which you can watch here. A company called Allegorithmic (which we'll get to below), did an amazing job creating two PBR Guides that you can find on their website here (For free!).

It is very important to understand how PBR works, as all the Real-Time Engines have fully adopted this workflow. When users are ready to become more advanced in Materials, the Real-Time Engines allow for custom shader development. This is a bit out of the scope for this class, but I do recommend looking this up when ready. It's some advanced work that deals with a lot of coding, but it will give you the freedom to really create some cool interesting Materials for the VR Experience!

Now these PBR materials are not easily exchangeable by themselves. For example, I'm not able to take a Material in Unity and bring it into Unreal. There is an alternative solution to all of this, and it is called Substance. A Substance is a special file created by a company called Allegorithmic. They are now the industry standard when it comes to materials, and pretty much every AAA Gaming Studio is now using their technology in their games. And anyone who has seen me talk before, has definitely heard me say their name in my presentations. All my Material work, either for High-End Raytracing or for Real-Time Engines, is done with their technology. This is mainly due to the procedural nature in which their application work. But also, more importantly, I can create a Material once and move it to any application I am working in! In terms of productivity in a pipeline, that is a huge time saver!

They offer three key platforms that will considerably help with the texturing process for Real-Time Engines:

Substance Designer

As the name implies, Substance Designer is the application that will design materials from scratch. By default, it will give users the option between the Metal/Roughness Workflow or the Specular/Glossiness Workflow. I recommend working in Metal/Roughness, but this is mainly dependent on which Real-Time Engine being used. Most the Real-time Engines support Metal/Roughness, but this can always be converted to Specular/Glossiness later if needed.

Pretty much any material that someone can think of, or cannot think of, can be created procedurally inside of Substance Designer. This dramatically opens the possibilities for the Visual Experience inside of Virtual Reality, because the sky is now the limit. Even though the new standard is called Physically Based Rendering, this does not mean the materials need to look Photo-Realistic. The terms tend to get synonymous due to how

they sound and are defined. Even stylized Materials can be Physically Based. This is where the style of Hyper-Realism comes into play. (Think Overwatch by Blizzard.)

Substance Designer also comes with an extremely powerful Texture Baking Tool. (We will cover Texture Baking below). Combine that with the ability to "Rig" Parameters that can be exposed at Run-Time, and Substance Designer becomes an extremely powerful tool to have in the bag.

Substance Painter

And just like Designer, Substance Painter sounds exactly like it is. This application will allow users to directly paint textures onto a 3D Model. (Which still needs to have its UVs unwrapped!) This kind of workflow blows the traditional method of having to paint textures by hand in a 2D Space, then view them later in 3D. Everything is now all done at once, and even multiple Map Types (Such as Normal, Roughness, etc) can all be affected at the same time as well! This dramatically speeds up production of texturing assets for Real-Time.

There are also additional tools inside of Substance Painter, such as their Particle Brushes. These can simulate certain effects that would be rather hard to paint by hand, such as water running down the side of an object, or creating random fractures. There are also other generators that can add additional wear and tear to objects, such as scratches, thumbprints, or peeling paint!

Bitmap2Material3

The last tool from Allegorithmic is the simplest in its form. Bitmap2Material 3 (B2M3) allows photographs of textures to be converted into PBR Materials. This little tool is extremely helpful when converting over materials that were not initially created in Substance to begin with. There are two flavors of B2M3. One is a stand-alone application, but the second is my favorite. Essentially B2M3 is a Substance File, and can be directly imported into Substance Designer or the Real-Time Engine itself! The latter is my favorite, as I can skip everything else and jump right into the Real-Time Engine. All the controls are the same as the Stand-Alone Application, but now I can just see everything in place as I'm working. It just simplifies the process enough to be extremely convenient to work this way.

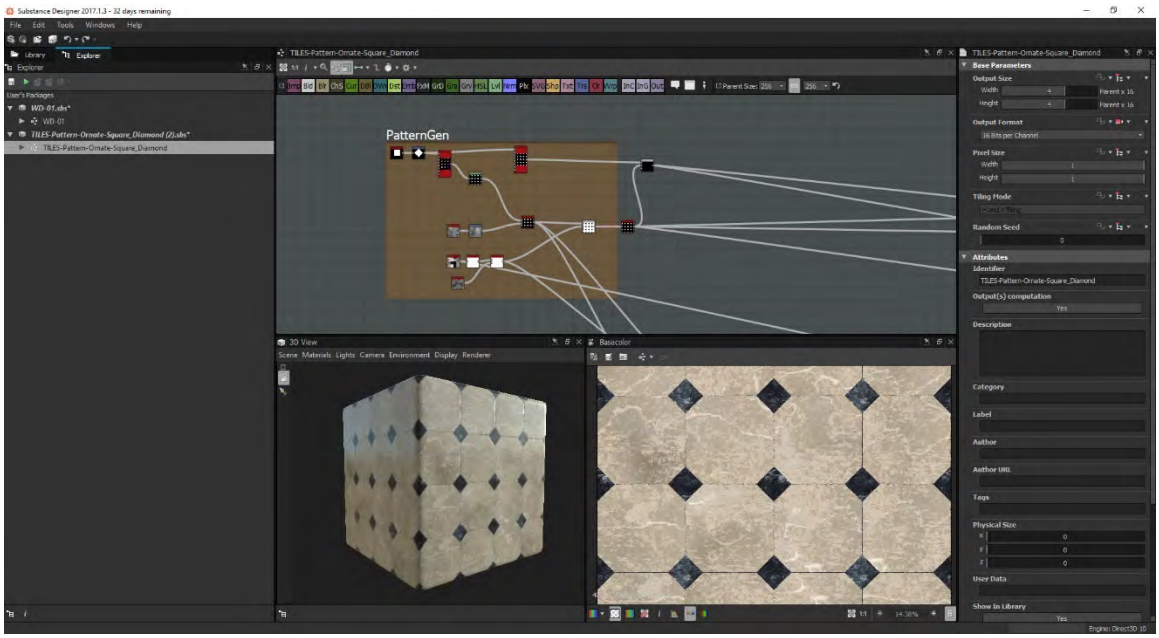
NOTE: There are plenty of other applications out there on the market to help with texturing if Substance does not fit into the pipeline. 3D Coat, XNormal, CrazyBump, Quixel, and Megascans are all good alternative sources for Materials in Computer Graphics, especially for Real-Time Engines. But for the purpose of this course, the rest of my texturing examples will be done with Substance.

Creating and Authoring Substance Files

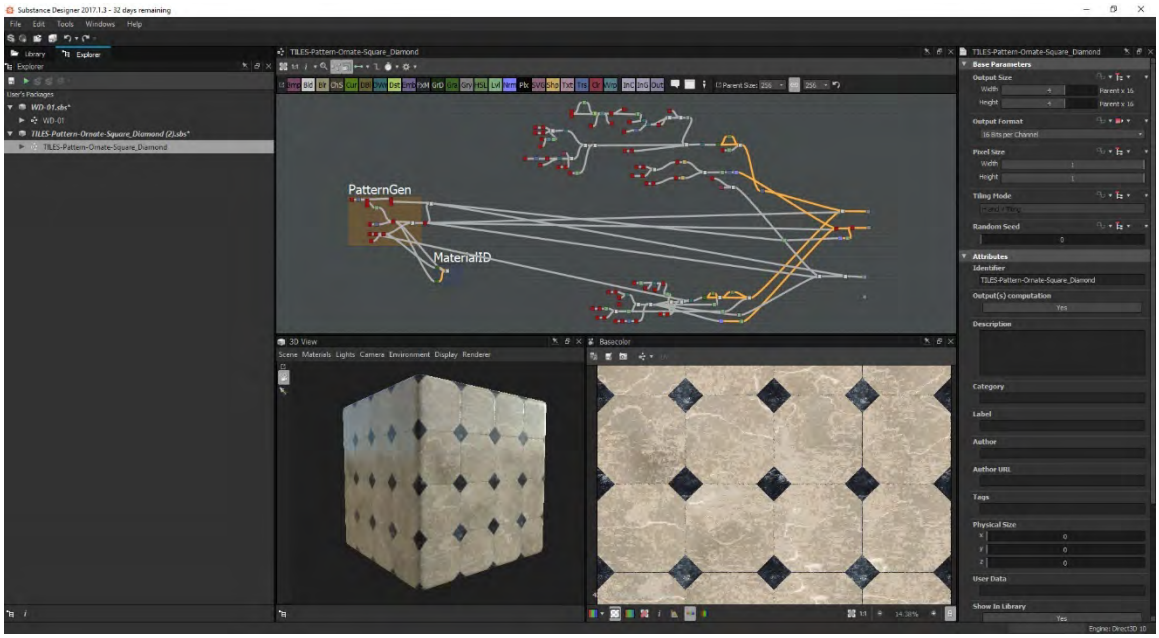
A whole day could be dedicated to texturing, let alone talking about Substance. In fact, Allegorithmic dedicates 3 days in the Summer to their own conference called Substance Days! Last year at Autodesk University 2016, I gave a course on getting started in Substance, and how to move from Substance Designer into a Raytrace Render Engine. That course can be viewed here. (There are also a ton of good training videos on Allegorithmic's YouTube Page.)

New users to Substance Designer might find it a little daunting, as the application can be intimidating. However, those who push through, and wrap their heads around it, will reap the benefits. The following are my guidelines when creating a substance from scratch:

1. **Big then Small**
   As with any piece of Art, you should always start with the big ideas and then work down to the small details. Making a Substance is the same way!
2. **Start with Shapes**
   Look at the material that is being generated. What are the biggest shapes there? Start with those, and then make the next set of shapes you see, and so on. Eventually you'll get down to the little details. All of this can be done through the Shape Node, or by blending multiple Nodes together! Experimentation is one of the best things about Substance. There are always Happy Accidents to be made. (Just remember to save those nodes for late, or roll with them!)

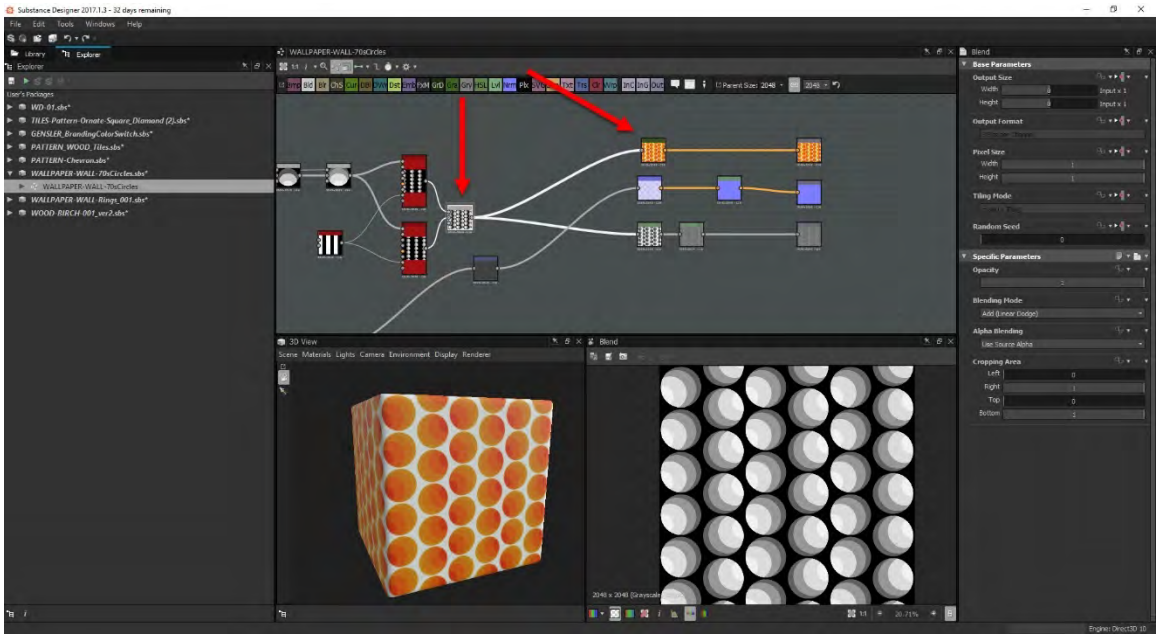


Notice the PatternGen Section highlighted here. This is where the over-all pattern for the tile floor starts. It controls the spacing, the diamond shapes, etc. Now if we zoom out…



Notice how complex the graph can become! And this is probably still a "simple" graph compared to others. However, the PatternGen Section is there all the way on the left, and is essentially the "Start" of the graph. Everything is derived from this one section.

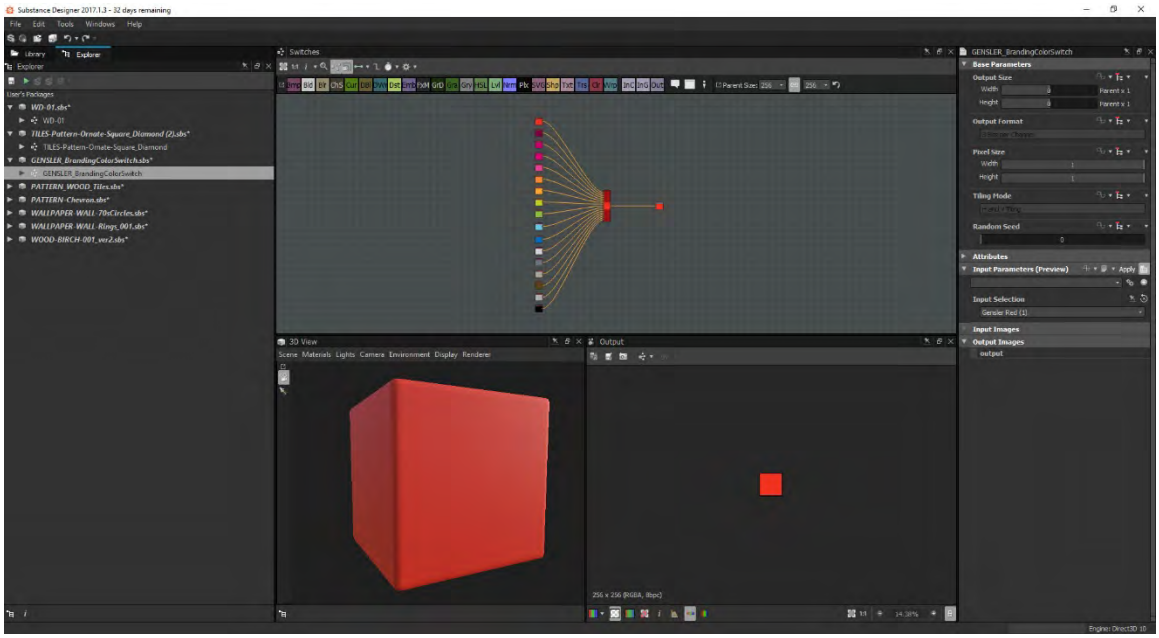
3. **Values Values Values**
   While creating the Substance, it is best to work in Grey Scale up until the end of the process. Not only does it help optimize the Substance Graph, but it also helps the over-all design process. If someone can tell what kind of material is being generated just from the greyscale, it will look amazing when color is added. This goes for everything else too. If you can't tell what is going on in a Photograph or a Rendering when it is greyscale, the values are off and need to be fixed!



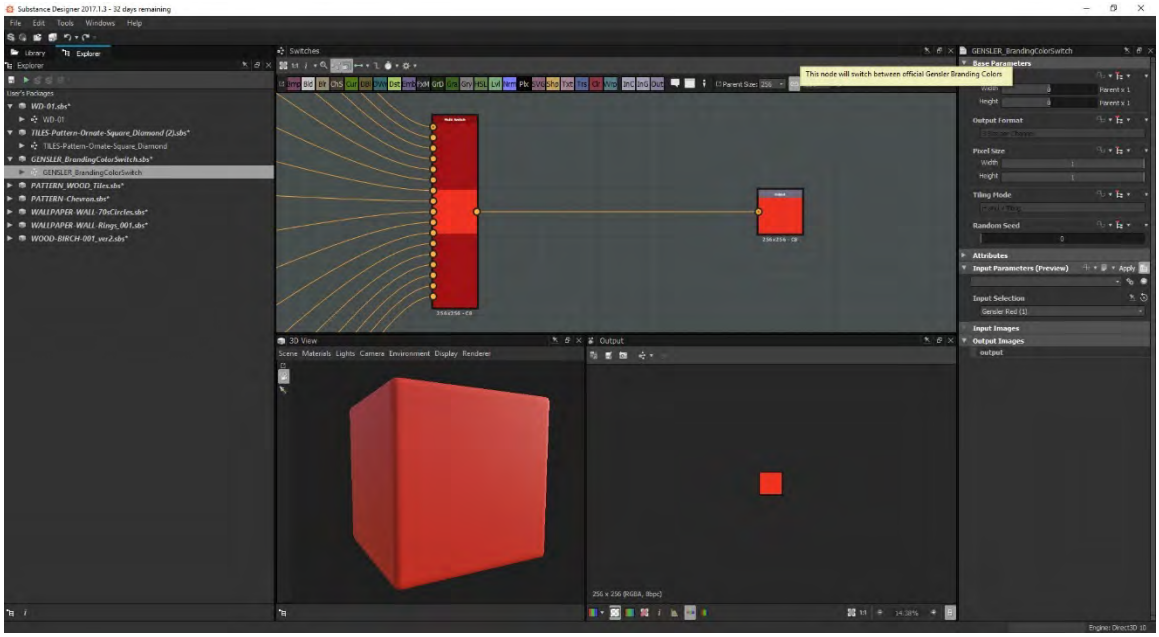
*Here is a good example how working in greyscale is important until the end. The texture is reading well up until the point to add color, which can be done by using the Gradient Map Node. The power here comes from the Gradient Map Node, as it can assign any color associated to any value in the greyscale input. Also, this now gives the ability to create multiple color options all based off the same greyscale input! This is **EXTREMELY** important when setting up a Substance for multiple options that include different colors. The Substance will perform faster overall than having to re-generate multiple color schemes independently.*

4. **Create Your Own Nodes**
   A powerful feature of Substance Designer is that every Node was made within the Application itself! And it gives users the ability to create their own Nodes from any set of Nodes. If there is something that is repeating over and over, or an interesting way to make something is found, create a Node out of them! This will help production of Substances speed up dramatically!

   This can be done by highlighting all the nodes being used, right-clicking, and then saying “Create Graph From Selection”. The only thing left then is to create an Input and Output Node for the new graph. This Graph can be published as a .sbsar file or just clicked and dragged into any other Substance Graph. (See Examples Below.)



This is a very simple example of what a Custom Node could look like. This is a “Gensler Branding Color” Node. I have set this up to give the user the ability to switch between all of our standard branding colors for the Firm. The user just has to drag and drop this node into their graph, select the color, and plug it in!



Here is a closer look at the node. It is mainly controlled by this massive MultiSwitch node on the left here. And then it connects into an Output Node, which is important. Custom Nodes will need an Output Node to connect to another Node in another graph. An Input Node could be added to a Custom Node to allow nodes from different graphs to be plugged into it as well! I did not need one in this case, but there are times where that will be important.

When the Substance is ready to be Published, it is just a matter of Right Clicking on the Package and selecting “Publish .sbsar File…”. This will prompt the user to pick a location for the file, and then a few other options to select. The default options are just fine, but do change them to your needs. Once the .sbsar file is saved, it just needs to be directly imported into any supported Real-Time Engine!

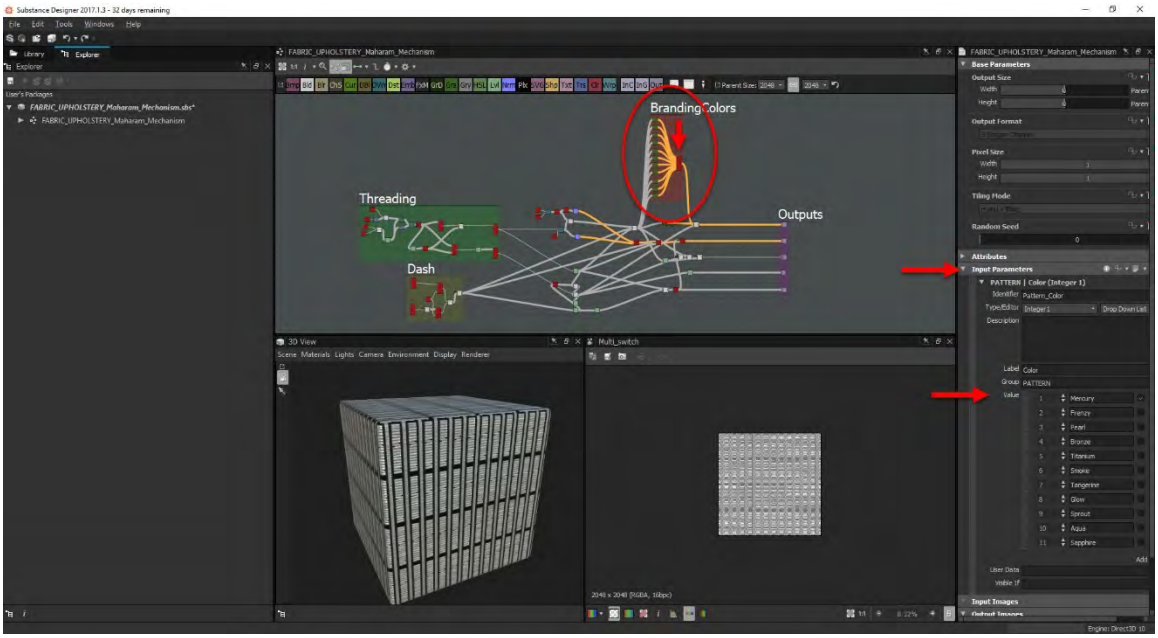
> NOTE: At the time of writing this document, Stingray/3ds Max Interactive does not support the direct importing of .sbsar files. Unity and Unreal however do support this feature. An additional Plug-In needs to be installed with Unreal, can be found for free on their Marketplace. For Unity, it works right out of the box.

“Rigging” Substance Files

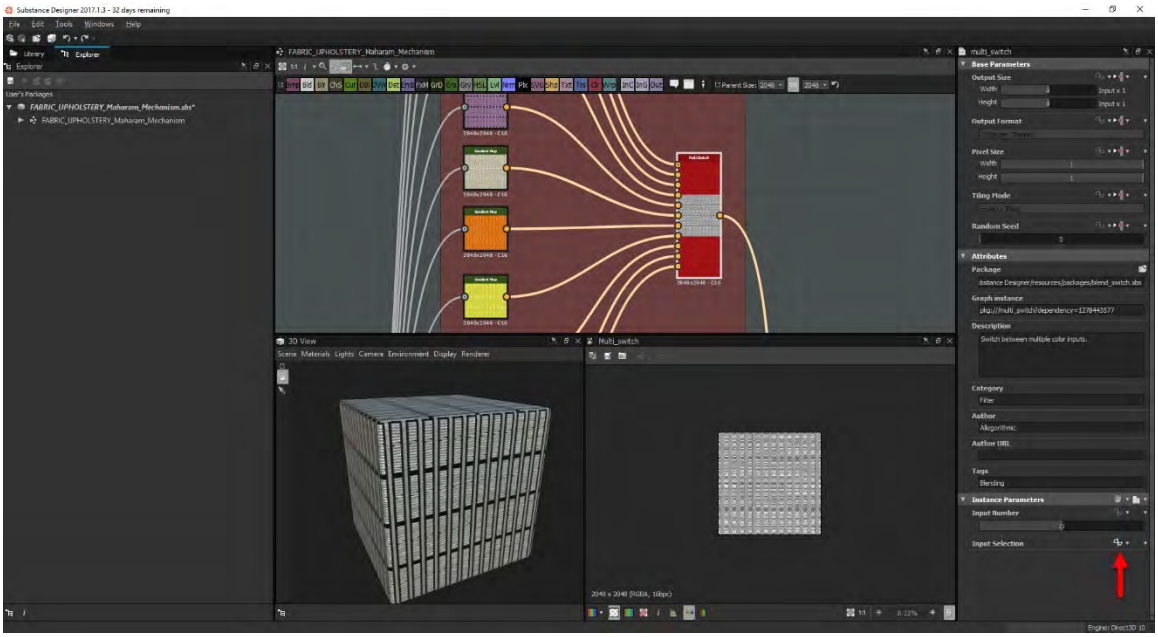
One of the power features about a Substance is that any parameter in a Graph can be exposed to the user. This can be just about anything: Color, Tile Count, Pattern, Shapes, Wet/Dry, Snow, etc. With these parameters exposed, it can be coded into the VR Experience! Either players can trigger certain effects, such as water spilling on a surface. Or players could change the color, or pattern, of a material through some kind of UX/UI Element that is designed. Working with Substances this way will greatly increase productivity and enhance the VR Experience even more to your liking!

Exposing a Parameter to the user is relatively simple:

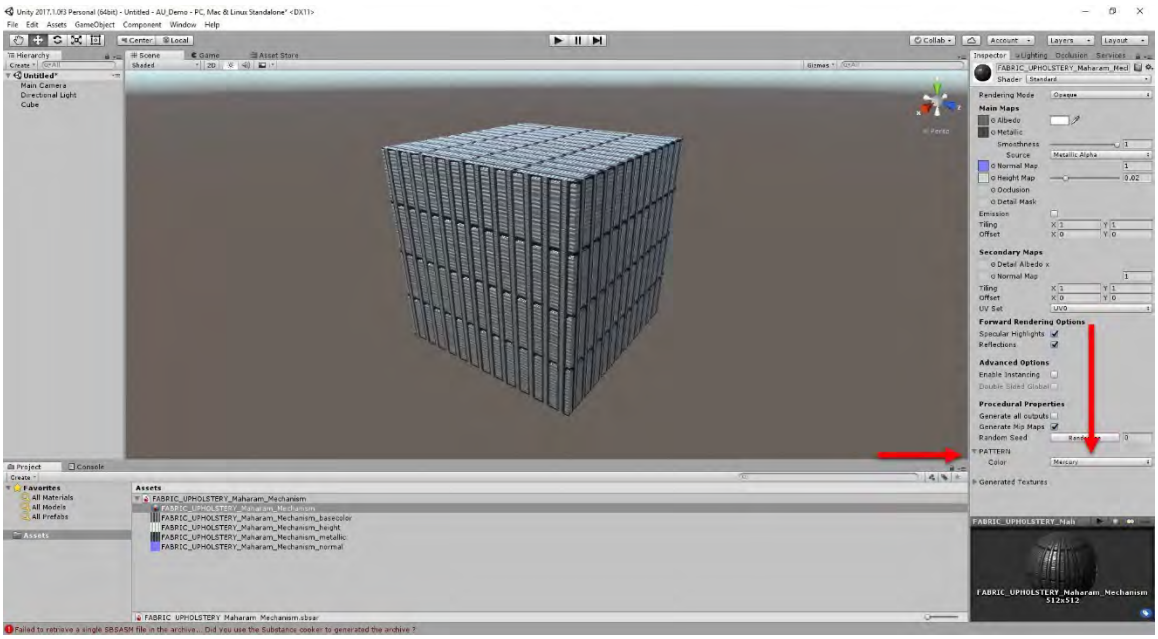
1. Select the Node that has the desired Parameter to Expose
2. Click the Function Icon on the Right
3. Select Expose from the dropdown list
4. Either leave the name parameter at default, or give it a new name using the dropdown list. Then click OK
5. Double Click into the Graph Background to bring up the Graph Properties
6. Under Input Parameters, find the exposed parameter that was just set
7. Customize the Parameter to the desired results. There are a lot of options here! Make sure to give it a Label and an Identifier
8. Expose any additional parameters with steps 1-7
9. Export the Package as a .sbsar file



This Substance has been set up to allow the different variations in color for this fabric. In its Input Properties section, I have exposed the MultiSwitch's "Input Selection". Then I renamed the different channels to match the branding color's naming.



If I dive down to the MultiSwitch Node. The highlighted icon shows that this Input for the MultiSwitch Node is Exposed. It is just a matter of clicking this Icon, and naming the exposed Input something unique. (Though the name could stay generic.)



Now if the Substance (.sbsar) file is imported into Unity, the exposed color selection is now in the Unity Editor! This can be scripted into a UI Element to allow a Player to swap the color of the fabric to any of the specific colors that was specified in the Substance!

Baking Materials

This will become typical for the pipeline if it isn't already. The idea behind Baking Materials is to remove stress off the GPU to increase performance. All kinds of information can be Baked for a Material. The biggest use-case for Baking is for Normal Maps from High-Poly to Low-Poly Objects. But also Lighting information is Baked, which can be done by the Real-Time Engine or externally by a Raytrace Engine. Even Ambient Occlusion can be Baked from a mesh. In this handout, we're going to focus on the former: Backing Normal Maps from High-Poly to Low-Poly Objects with Substance.

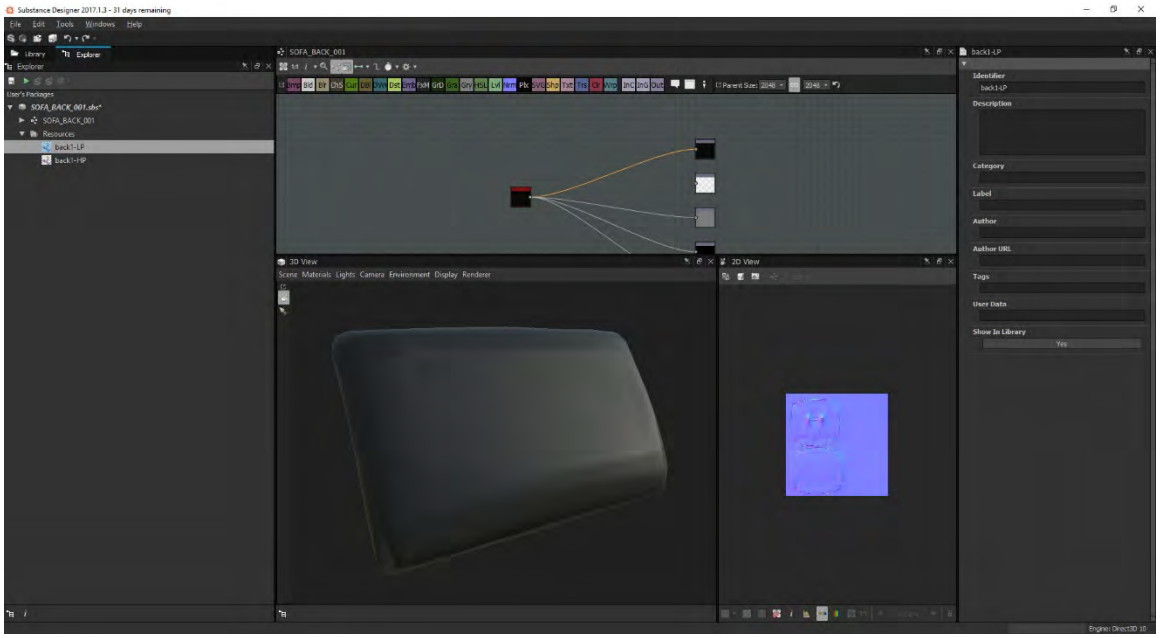
> **NOTE:** Now this can be done a with several other applications, including 3ds Max itself! But since I use Substance in my Pipeline, and its Baker is good, we're going to focus on that workflow. But just note this is not the only way to handle the baking of Normals.

The purpose of baking Normal Maps from a High-Poly Object is to capture that data and project it onto a Low-Poly Surface. The ending result will the Low-Poly Object looking like the High-Poly Object! (Remember the more polygons on screen result in slower rendering times for the GPU, and thus creating the effect of motion sickness!) So this is a beautiful "workaround" for that problem. Normal Maps are by no means new to CG or the Real-Time Engines. Games have been using them for a better part of a decade now.

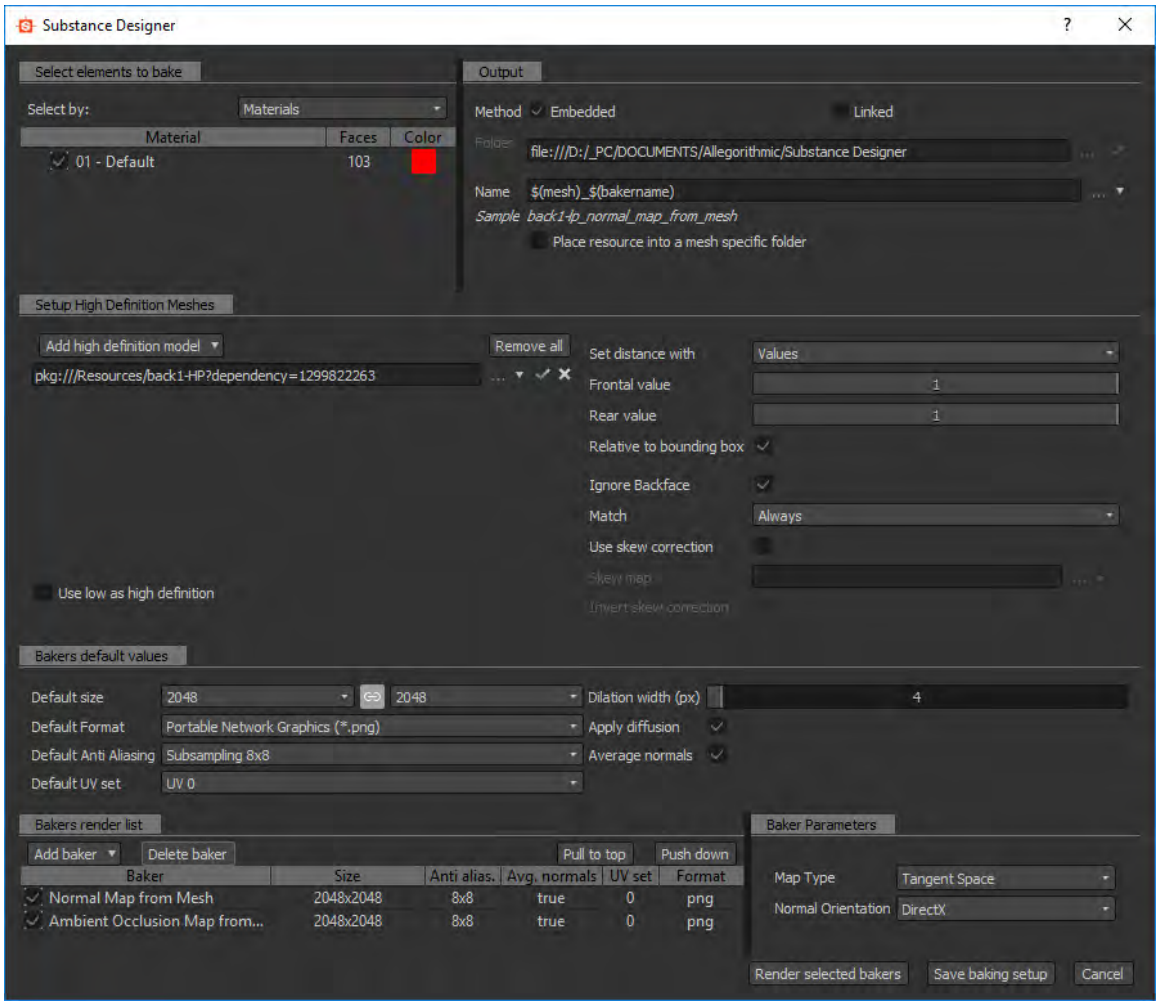
To bake the Normal Maps (or any other Maps from a HP to LP Mesh) do the following:

1. Make sure the HP and LP Mesh are right on top of each other in 3ds Max.

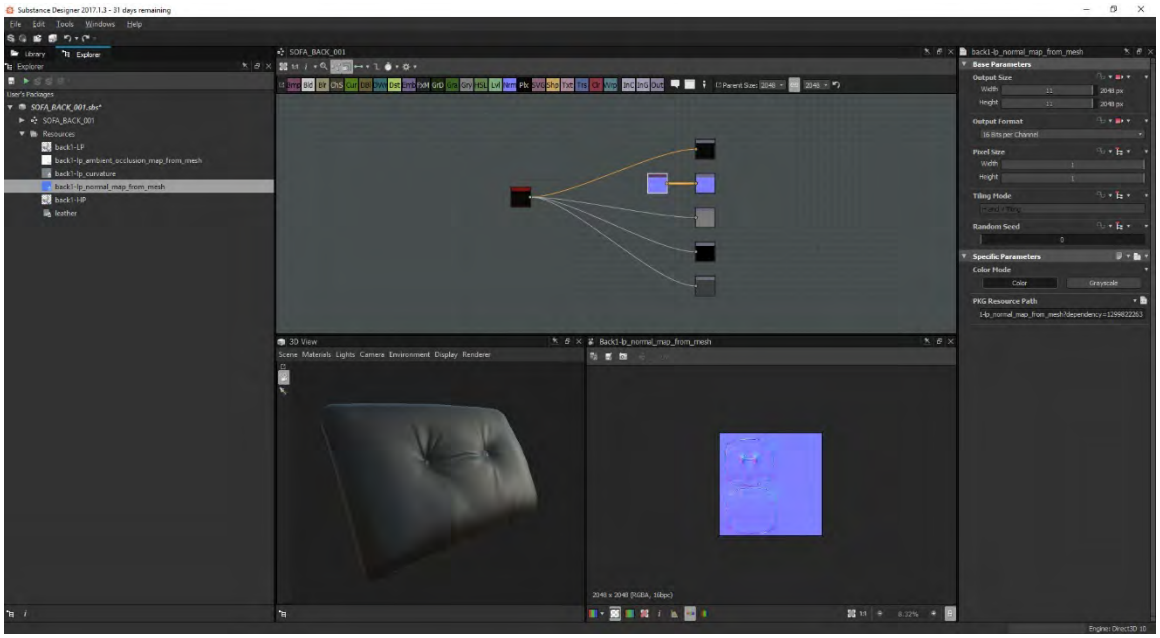
2. Export out each into their own FBX File with a naming convention. I recommend adding “_hp” and “_lp” to help tell them apart.
3. In Substance Designer, start a new Package and Graph.
4. Click and Drag these two FBX Files on top of the newly created Package.
5. A new “Resource” Folder should be listed under the Package. Open that and highlight the Low Poly Model
6. Right-Click on the Low Poly Model and select “Bake Model Information”
7. A new window will open with all kinds of options.
    a. I recommend learning what they all do, but for the moment we will do the following to produce the result we are looking for.
8. Under the “Setup High Definition Meshes” section, click on the “Add high definition model” and select “From Resource”.
9. Select the High-Poly Model from the list
10. In the “Output” section, select either Embedded or Linked.
    a. This will either put the Normal Map directly into the Substance, or link it in by writing it to disk at a specified location.
        i. My preference is Embed, but this is my own personal preference. I do however highly recommend Embedding when creating an SVG Map from UVs.
11. Under the “Bakers Default Values”, set the desired resolution, file format, anti-aliasing, and UV Set.
12. Under Bakers Render List, click the “Add Baker” button. Select “Normal Map from Mesh”
    a. Add any other maps that are needed.
13. For the Normal Map, under the Baker Parameters, leave this to Tangent Space. Under Normal Orientation select the desired Target: DirectX or OpenGL.
    a. Leave it at DirectX if you are unsure.
14. Click the “Render Selected Bakers” button at the bottom right corner to bake all the texture maps.
15. Once the Bake is complete, close the Baker Options Window
16. Click and Drag the “Low Poly” mesh into the 3D Viewport to add it in as a display object.
17. Click and Drag the newly created Normal Map into the 3D Viewport, and select Normal from the dropdown list that appears.
18. If there are artifacts in the Normal that look like it didn’t capture enough detail, the Normal will need to be re-baked.
19. Right-Click on the “Low Poly” mesh in the Explorer and select “Bake Model Information”
20. Under the “Setup High Definition Meshes” section, increase the “Frontal Value” and “Rear Value”.
    a. This might take some experimenting. I personally like to crank this up to 1 at this point for each, and work my way back. But 9/10 times a value of 1 has always given me what I needed.
21. Click “Render Selected Bakers”
22. The Normal Map should update on in the 3D Viewport as soon as the bake is completed.



Here's the Re-Topologized Pillow from the previous example. It's linked into Substance Designer, along with the High Poly version. To bake the Normal Map, right click on the Low Poly object in the Explorer.



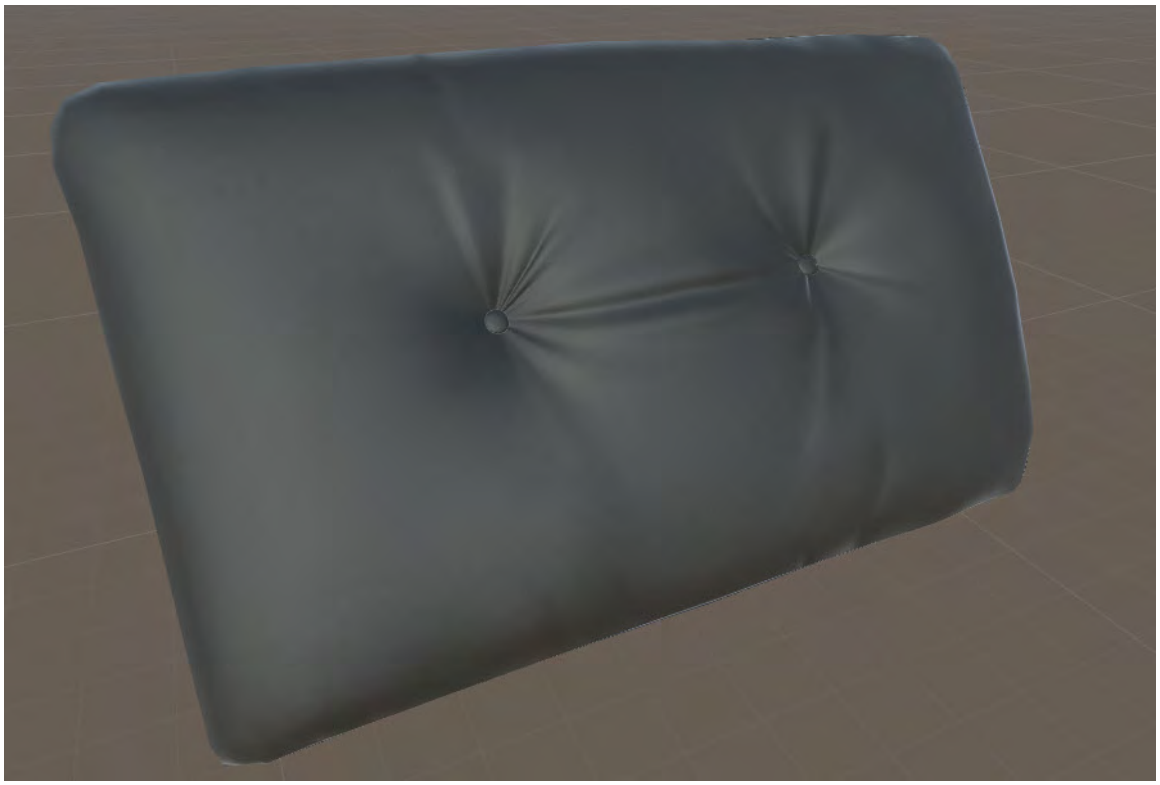
The Bake Window will appear! This is where the High Poly Mesh can be loaded in for the reference, and the specific texture maps for Baking can be generated. In this case, we're doing Normal Map. But I've also included an Ambient Occlusion for fun. When ready click the Render Selected Bakers button at the bottom right.



The newly generated Normal Map can be connected to the graph, and displayed in the 3D View!

For reference, this is what the High Poly mesh looks like…

And this is the Low Poly mesh with the Normal Map! It's nearly identical!



And just for kicks, here is the Low Poly mesh + the Substance loaded inside of Unity!

Real-Time Engines

This is where things get interesting. There is a lot of technology advancements happening at a rapid pace in the Real-Time Engine scene right now, and it will probably be further along by the time you read this at AU. While there are several Real-Time Engines out there, there are probably three that are at the top right now:

- **Unreal**
- **Unity**
- **Stingray (3ds Max Interactive)**

Now some of you are probably thinking “Stingray!?” But hear me out. Autodesk has done a substantial amount of work brining Stingray up to par. It still has a lot of work to be done, but considering where it was this time last year, it is very impressive where it stands right now. With all of that taken into consideration, plus the interoperability between 3ds Max 2018 (And that it COMES WITH 3ds Max 2018), Stingray deserves some serious consideration here. I’ve been personally excited to play around with it recently, and even more so to see where Autodesk does take it. The future looks bright for Stingray/3ds Max Interactive.

The question I get asked the most is “Which one do I use!?” or “Which one is the best!?”…

Well the simple answer here is:

They all are!

The not so simple answer is:

This really comes down to your preference, project needs, development support, and budget. The best way I can describe these three Real-Time Engines is to compare them to 3ds Max, Maya, and XSI (too soon?). Now I’m not going to tell you which one I think is which, but the point here is that they all do essentially the same thing.

They will all:

- Take your FBX from 3ds Max
- Use PBR Materials
- Have Light Baking Engines for Static and Dynamic Real-Time Lighting
- Use a coding language for custom scripting
- Have an Asset Store
- Publish to any platform
- Support any HMD

Of course, we could dive down deep into who has the best lighting, or it’s easier to code in C# than C++ or Lua, but those come down to personal preference. Because any of these Real-Time Engines will do what you want.

The biggest factor in picking which Real-Time Engine to use will most likely be your Budget. These Real-Time Engines all have something in their EULA that makes them “Free”. But let’s be real honest, nothing is Free in this world. So, make sure that the cost

of the software with their EULA meets the requirements of the project that is being generated. There is no simple answer to this, and will be heavily dependent on each individual company and project.

With all that said, the best thing to do is: **PICK ONE!**

Just get in and go. Start making stuff and experimenting! The overall workflow is relatively the same once you move into a Real-Time Engine. Again, compare them to 3ds Max, Maya, and XSI (It will always be too soon…). If I wanted to model a House in 3ds Max, I could model the same one in Maya. It is just the tools to do it in Maya would be just a little different from the tools in 3ds Max. The same goes when working inside Unity, Unreal, or 3ds Max Interactive (Stingray).

Now there a few technical things, which should be noted, to help with Performance in the VR Experience. Each of the three Real-Time Engines discussed above can handle these next two sections in their own way. So, I recommend looking at their documentation on how to enable these features.

Occlusion Culling
This one is a BIG feature that needs to be enabled from the start. Occlusion Culling will add additional boundaries to each object. These boundaries will determine if the object is rendered or not when the camera is looking at the object. Meaning, if the object is not in view, it isn't loaded! This DRAMATICALLY increases performance in Virtual Reality as the GPU is not loading or thinking about objects that cannot be seen.

Now each Real-Time Engine handles this a little differently, but the core concepts are the same. The following links are for the documentation on each Real-Time Engine's Occlusion Culling Methods.

Unity – https://docs.unity3d.com/Manual/OcclusionCulling.html

Unreal – Not much official documentations specifically on this. However, Occlusion is enabled by default. This can be found in the Project Settings > Rendering Tab.

3ds Max Interactive (Stingray) - https://help.autodesk.com/view/Stingray/ENU/?guid=__stingray_help_building_levels_create_game_objects_create_occluders_html

On the left is the Unity Scene View, and on the right, is the Game View. Notice how it looks like half the model is missing in the Unity Scene View. This is because the Camera is not facing anything in that direction, so the geometry is not loaded!

Level of Detail (LOD) Mesh

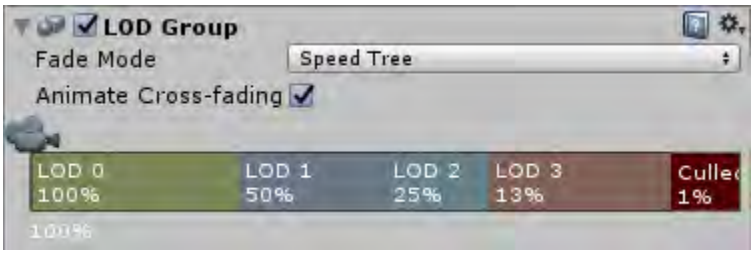
Another big help for Performance inside of a Real-Time Engine is to set up LODs for the meshes. This will allow the Real-Time Engine to lower the quality of the object, on-demand, depending on how close the Player is to said object. This is SUPER helpful for complex objects such as Trees, Furniture, Characters, or even some smaller entourage. Because again, the less polygons on screen, the less work the GPU has to do. It doesn't make sense to have a 20k poly chair that is 20 feet away from the Player. The Player will never be able to make out the detail being shown in the 20k poly chair from that distance. So why not have the chair be 5k polys at that distance?

Each Real-Time Engine handles LOD Meshes a little differently, but it all works the same way in the end:

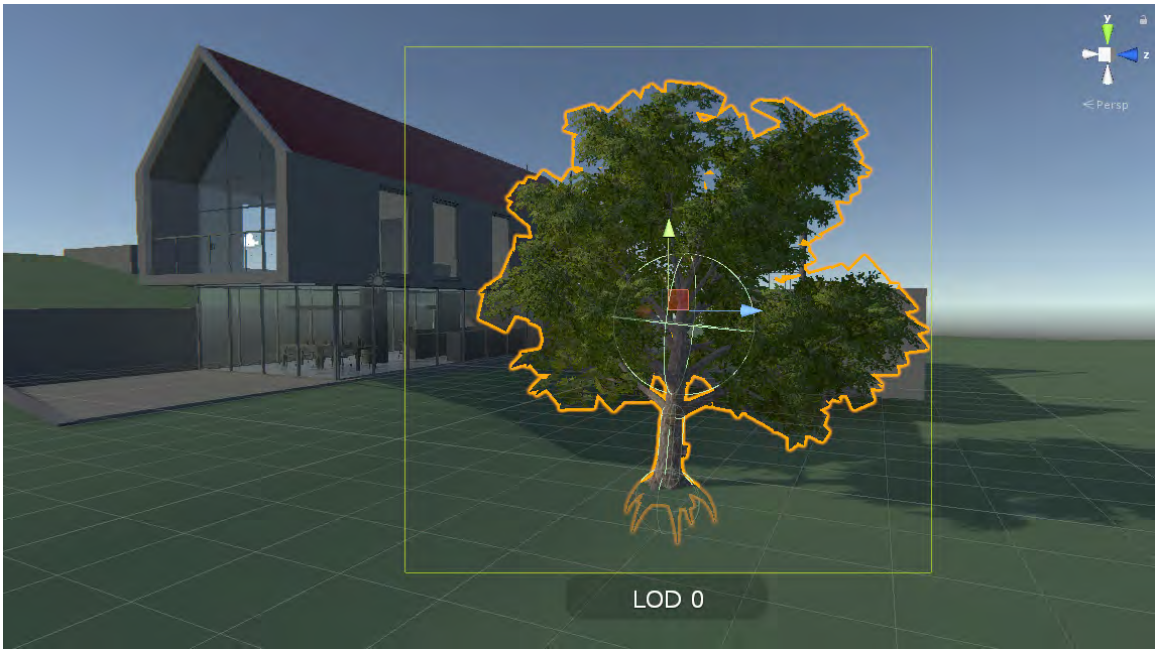
Unity - https://docs.unity3d.com/Manual/LevelOfDetail.html

Unreal - https://docs.unrealengine.com/latest/INT/Engine/Content/Types/StaticMeshes/HowTo/LODs/

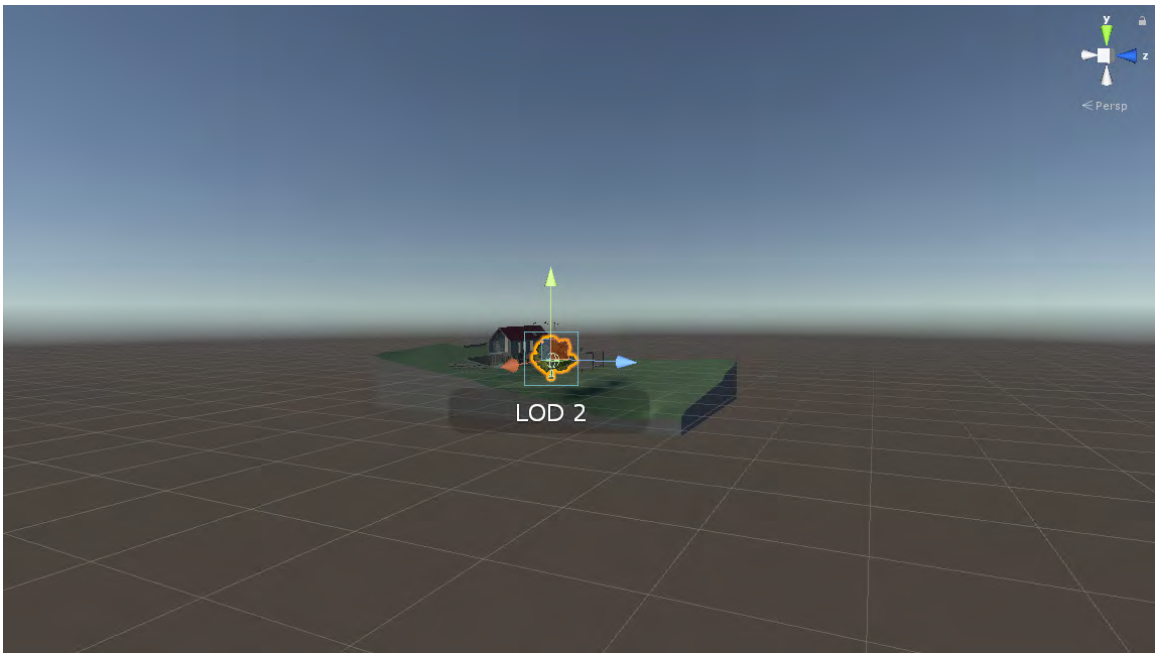
3ds Max Interactive (Stingray) - https://help.autodesk.com/view/Stingray/ENU/?guid=__stingray_help_importing_assets_level_of_detail_html



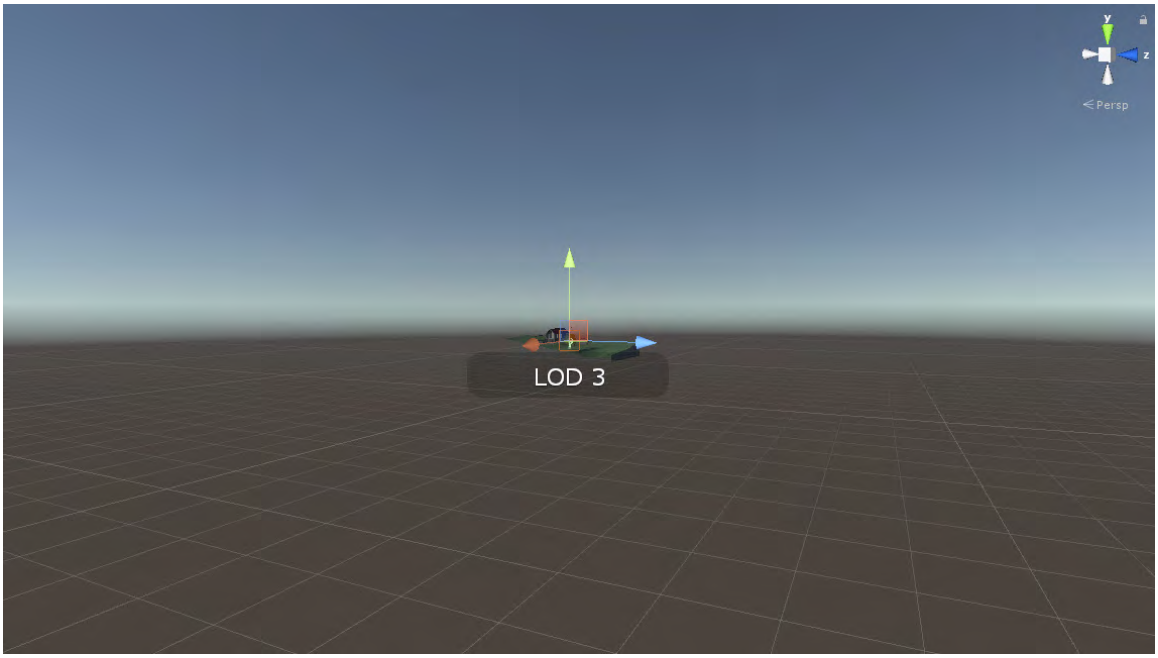
This Component in Unity controls the Distance for LOD Models…



When super close, we see that the tree is at LOD0, which is the highest detailed version of this tree. And as the player steps back…



Now we can see LOD2 has been loaded…



And now LOD3 has been loaded. This might be "too far" for this particular scene, but that's ok as the designer can change this in the LOD Component.

(These examples were using the free SpeedTree Assets from the Unity Asset Store.)

Mip Maps
These are just like LOD Meshes, but they are designed for Textures! Mip Maps are various resolutions of the same texture that get loaded on-demand depending on the distance from the camera. Just one more thing to take the load off the GPU!

Each Real-Time Engine handles Mip Maps a little differently, but thankfully all of them have this feature on by Default! Any adjustments are relatively simple to set up in the Texture Editors in each Engine.

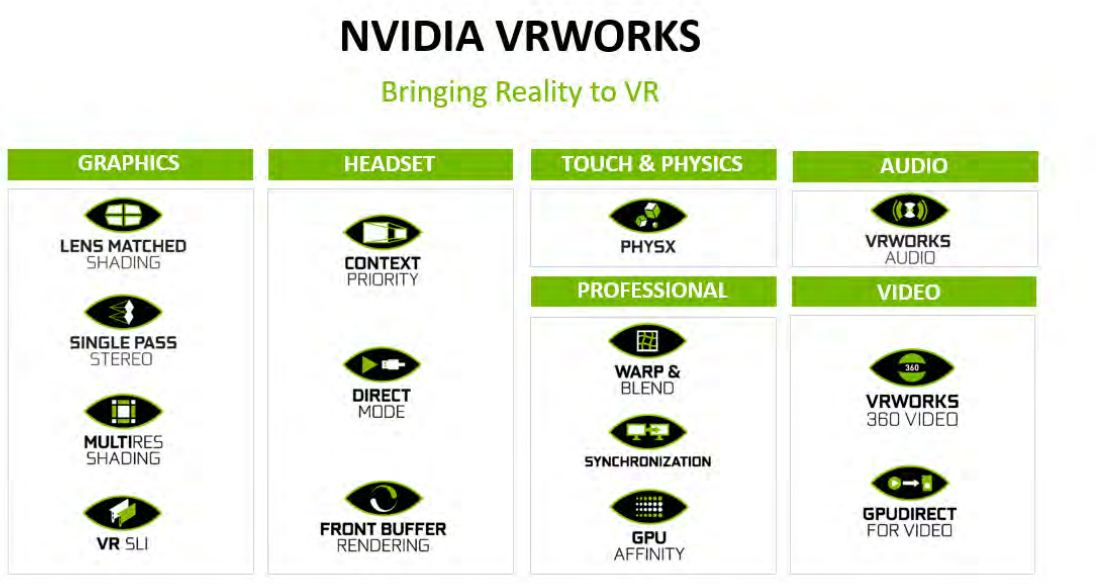
Unity - https://docs.unity3d.com/Manual/class-TextureImporter.html

Unreal - https://docs.unrealengine.com/latest/INT/Engine/Content/Types/Textures/Properties/

3ds Max Interactive (Stingray) - http://help.autodesk.com/view/Stingray/ENU/?guid=__stingray_help_lighting_rendering_shading_and_materials_work_with_textures_texture_manager_html

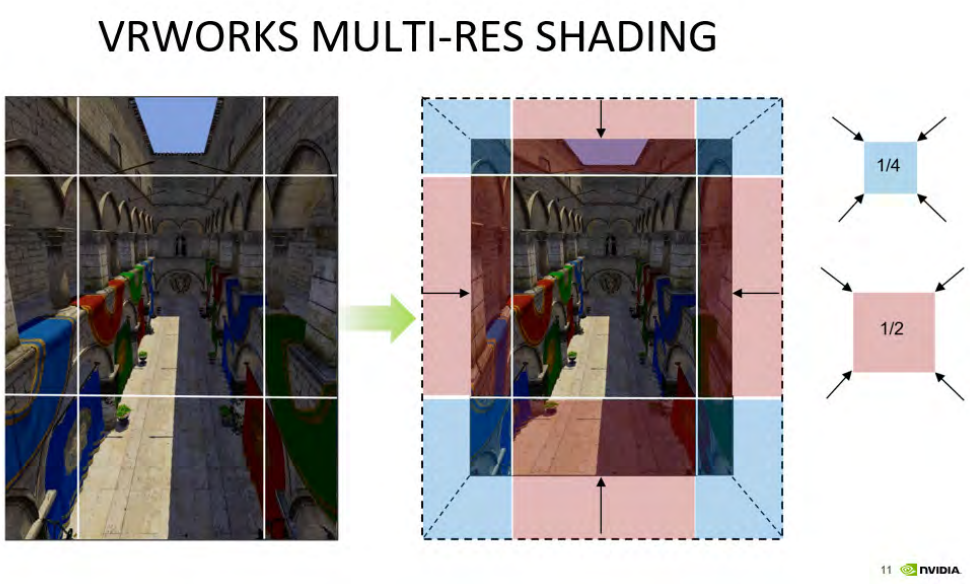
NVIDIA VRWorks
This is something to pay close attention to, and add to your VR Projects when working with any of the Real-Time Engines. NVIDIA has put a lot of work into its VRWorks SDK to improve overall performance when working with Virtual Reality. A massive performance increase can be had when utilizing VRWorks in your project. And the best part is, if someone doesn't have a compatible GPU, the VR Experience will still play (just without the VRWorks SDK).

(Thanks to NVIDIA for providing the graphics for this section.)



Multi-Res Shading (MRS)
The Real-Time Engines adds a post-processing distort the Left and Right eyes to offset the distortion of the HMD. And as a result, we end up rendering too many pixels around the edges of each frame. This is where Multi-Res Shading comes into play. Each Eye is divided up into a 3x3 grid. The Center Grid is rendered at full resolution, but the perimeter grids are rendered at ½ and 1/4th resolution. This approximates the distortion that will eventually be added by the Real-Time Engine. And at this point, it means the GPU is rendering less pixels in the end. Less pixels equals faster rendering!

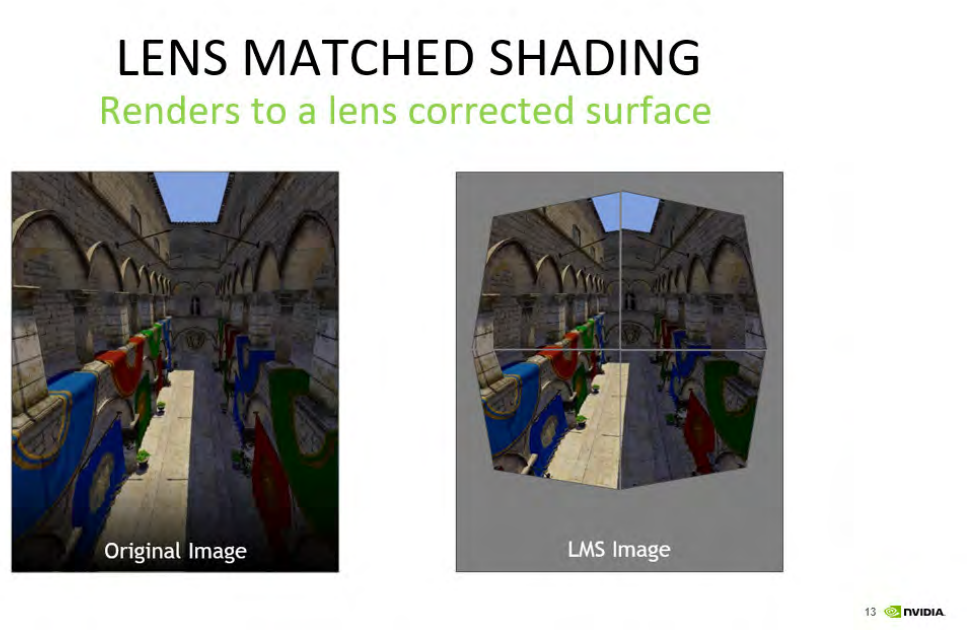
> **NOTE:** This feature is only available on NVIDIA Maxwell and Pascal GPUs. (GTX 900+ and Quadro M5000+)



This is a breakdown of how the MRS breaks a viewport into multiple resolutions.

Lens Matched Shading (LMS)
This feature uses new technology that can only be found in the NVIDIA Pascal GPUs. Using the new Simultaneous Multi-Projection technology, Lens Match Shading will render the view to a surface that closely approximates the lens corrected image that would eventually be sent to the HMD. The result ends up rendering less pixels that would typically be wasted on the GPU. Less Pixels equals faster rendering!

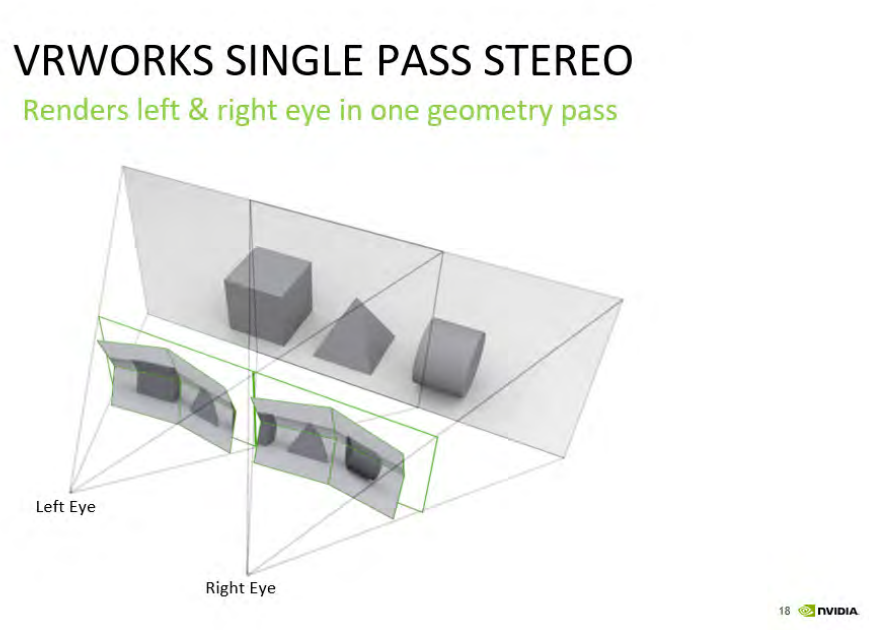
> **NOTE:** This feature is only available on NVIDIA Pascal GPUs. (GTX 1060+ and Quadro P4000+)



The LMS changes the viewport into something that matches the HMD.

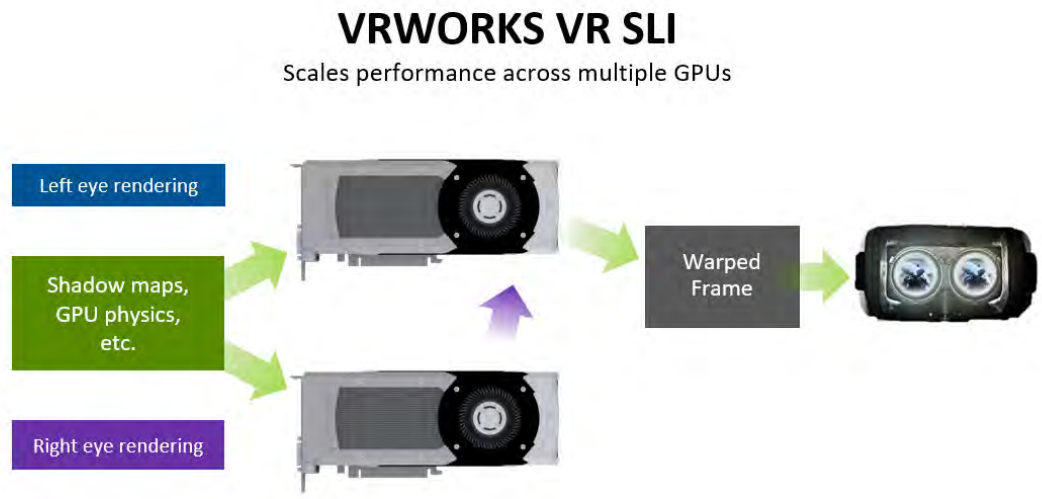
Single Pass Stereo (SPS)
Just like the Lens Matched Shading, this feature uses the new Simultaneous Multi-Projection technology that is only in the NVIDIA Pascal GPUs. Before adding this feature to a VR Experience, the GPU would need to render all the geometry twice. It does it once for the Left Eye, and another time for the Right Eye. The Single Pass Stereo eliminates this, and only has the GPU render the scene once! It then will project the scene to the left and right eye. This will allow developers to double the complexity of the scene, which results in more richer experiences!

> **NOTE:** This feature is only available on NVIDIA Pascal GPUs. (GTX 1060+ and Quadro P4000+)



VR SLI
Since each eye in are independent of each other, they can be divided to individual GPUs! This API will allow the rendering of each eye to happen on their own GPU. (GPU0 = Left GPU1 = Right) And then the final image is combined before being sent to the HMD. And the best part is this can be extended past just 2 GPUs! The VR SLI API does this by enabling GPU Affinity Masking. This allows the developer to determine which sets of Draw Calls go on each GPU. A secondary technology goes along with GPU Affinity Masking called Broadcasting. Without Broadcasting, the CPU in the machine still has to process for both eyes. Broadcasting fixes this by rendering both eyes with a single set of Draw Calls. This cuts the processing of the Draw Calls per frame by half on the GPU and CPU by half!

> **NOTE:** This feature is only available on NVIDIA Maxwell and Pascal GPUs. (GTX 900+ and Quadro M5000+)



VRWorks Audio
Visual Realism is not the only thing to consider when designing a VR Experience. Audio plays a major factor in the VR Experience as well (Something we haven't covered in this document yet.) The NVIDIA VR Works Audio helps simulate how sound really works inside of an environment. Using the NVIDIA Optix Ray Tracing Technology (The same that is used for visuals!), the VRWorks Audio will trace the paths of sounds in real-time. This will deliver a physically realistic result of how sound bounces off objects in the environment.

The VRWorks Audio creates the audio solution in real-time without any precomputed filters. The scene will load, and the acoustic model will be

generated on-the-fly. This gives designers unprecedented flexibility in designing soundscapes for VR Experiences!

Key Features in VRWorks Audio:

- Sound Propagation
- Occlusion
- Directionally/HRTF (Head Related Transfer Function)
- Attenuation
- Approximate direct path diffraction
- Material reflection, absorption, and transmission

NVIDIA has plenty of documentation around this, and each Real-Time Engine handles the implementation a little differently. What will need to be done is to register a Developer Account on the NVIDIA Developer's Website.

Unity
Developers will need to be running the latest version of Unity 2017.1 to deploy the VRWorks SDK. This can be added to the Unity project from the Unity Asset Store.
NVIDIA's Documentation can be found on their developer's website.

Unreal
Developers will need to create an Unreal Account, along with a Github account, and then associate the two accounts together from the Unreal Account Settings. Then the Unreal Engine Source Code will become available to the developer on GitHub. The following branches for the NVIDIA VRWorks SDKs will become available as well. The links for the branches can be found on the NVIDIA Developer's Website.

Stingray
Stingray supports a few features of VRWorks right out of the box in the latest 1.9 release! Supported features are Single Pass Stereo and VR SLI. Lens Matched Shading is being worked on. To enable these features, users will need to open the project's settings.ini file. In that file users will need to set the following lines accordingly:

- nv_single_pass_stereo_enabled = true
- nv_vr_sli_enabled = true
- disable_implicit_sli = true

NOTE: 3ds Max Interactive is currently running Stingray 1.8, so these features are currently not available. There should be a future update to 3ds Max Interactive that will bring it up to par with Stingray 1.9, or to the latest release at that time. I cannot be 100% sure of this as I am not involved with any of this development.

Audio

Just like Animations, Film, and YouTube Unboxing Videos, Audio can be the one thing that can make or break the experience. It will be imperative to make sure that audio is clear and sharp. Recording your own audio can be a bit challenging. So, if that is something that is required, I **HIGHLY** recommend investing into quality recording equipment.

Most people's first reaction is not to spend a lot of money on this equipment, but cheap equipment will literally sound like cheap equipment. Though that's not to say that there isn't some good affordable equipment here. For instance, I use the **Blue Yeti USB Microphone - Blackout Edition.** This microphone runs roughly $120, and has produced pretty good recording for some voice work I've done. (I also recommend getting a Wind Screen for the Mic as well.) For recording audio, I end up using Adobe Audition CC. I haven't done too much research around a lot of software for recording audio, but since Audition comes with a CC Subscription, I might as well take advantage of it. Not to mention how it syncs up with the rest of the Adobe applications!

Hiring professional voice-actors, or purchasing asset libraries for sound FX, is probably an easier route to go for audio. The biggest thing to note here is the licensing around the Audio. Third Parties who create audio for Professional Productions usually have different levels of licensing depending on how the media is being distributed. A few of the Real-Time Engines have a lot of audio that can be purchased for this purpose, so it is probably a good idea to start there!

Each Real-Time Engine handles audio a little differently, and their documentation can be referenced here:

Unity - https://docs.unity3d.com/Manual/Audio.html

Unreal - https://docs.unrealengine.com/latest/INT/Engine/Audio/

Stingray -
http://help.autodesk.com/view/Stingray/ENU/?guid=__stingray_help_working_with_audio_html

Other VR Tips and Ticks

There are a few other things to note about working inside of VR that are not covered by any technical piece of software.

The camera should be placed where someone will experience something!
We are so used to consuming media by looking at a screen, we automatically forget that is not what we are doing in VR. The essence of VR is to have someone *experience* something. Now we can feel emotion from a well-crafted 2D story, but once you put

someone inside that story… the game changes. VR Works the best when the user feels like the are a part of the story, and not removed and viewing it. This is the KEY difference between 2D Media and VR.

Don't place the View on top of anything un-normal
The quickest way to break the experience is to put someone on top of something they normally wouldn't stand upon. A Chair, or Table, is a perfect example of this. People tend to get freaked out because they know they aren't standing on a chair, but it *feels* like they are standing on a chair. Unless the experience calls for this, try to avoid it at all costs.

Place the Camera at 5'6" (For 360 Renderings Only)
The average height of someone is roughly 5'6". Most people can make up the difference in their head when looking at 360 Renderings. However, it the camera is moved any higher or lower than this, it can break the experience for anyone who isn't accustomed to that higher/lower height. This recommended value has proven to be the best in a lot of studies that I have done.

Be careful with Objects too close to the Viewer's face. (Especially in 360 Renderings)
An easy way to break someone out of a VR Experience is putting something too close to their face, or personal space. It again goes back to the braining known that something isn't really there, but it can see something is really there. Most viewers tend to freak out a little bit, and you'll never get them back into the experience after this. Unless you're going for a shock and awe moment, it is best to avoid this as much as possible. It is also extremely important NOT to do this for 360 Renderings. In more immersive VR Experiences, it is easier to get away with it as the user can have 6-Degrees of Freedom to move around.

The Challenge

Hopefully I have inspired some of you reading this document. I want to challenge anyone reading this, including myself, to try to create something more than a Tech Demo in VR. This new medium lends itself so well to Experience, it would be a shame to continue wasting it on the ability to just "Walk Around" in VR. There's so much more that the Design Industry can bring to this Technology, and so much we can learn from others! One of the most exciting things is every other Industry is still trying to figure this technology out. And we are in a prime position to not only figure it out, but to also make it look good!