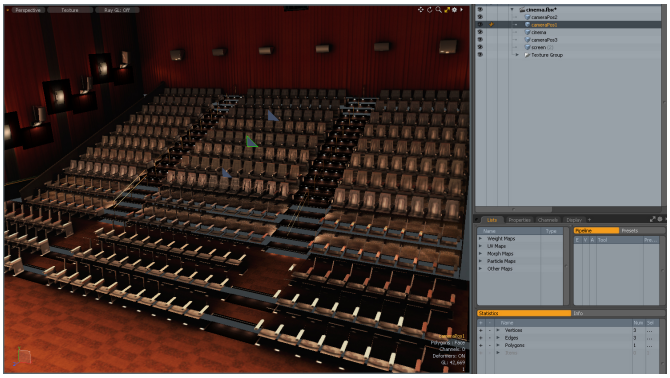
How to Create and Compile a Movie Theater FBX

v. 0.4



The cinema scene in MODO and its item names.

Steps to Create a Theater:

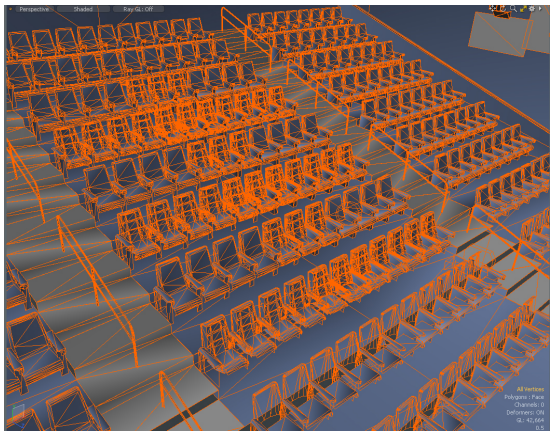
1. **Create a theater mesh.**
2. **Create a screen mesh.**
3. **Create two theater textures (one with the lights on and one with the lights off).**
4. **Create meshes that define the camera/view positions.**
5. **Create meshes that use an additive material (optional, for dim lights).**
6. **Run the FBX converter tool.**

Detailed Instructions for Each Step

1) Create a theater mesh

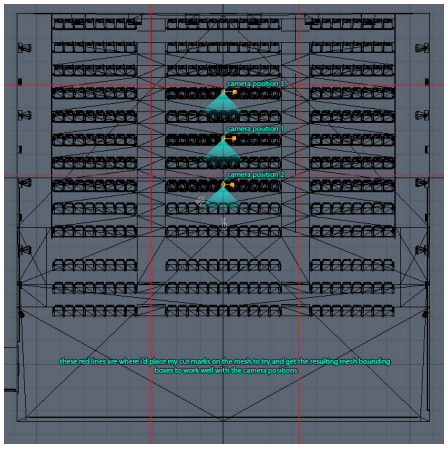
- Nothing special here. Just create a theater mesh, but here are some tips for how to optimize the mesh. Keep the polycount as low as physically possible as a rule. Rendering is a taxing operation, and savings in this area will benefit your entire project. As a point of reference, the "cinema" theater mesh has a polycount of 42,000 tris. Review *Performance Guidelines* for detailed information about per-scene polycount targets.
- Polycount optimization can be executed in a variety of ways. A good example is illustrated with the “cinema” mesh included in the SDK source files for Oculus Cinema. When you load the source mesh, you’ll notice that all the chairs near player positions are high poly while chairs in the distance are low poly. It is important to identify and fix any visual faceting caused by this optimization by placing cameras at defined seating positions in your modeling tool. If certain aspects of the scene look “low poly” then add

new vertices in targeted areas of those silhouettes to give the impression that everything in the scene is higher poly than it actually is. This process takes a bit of effort, but it is definitely a big win in terms of visual quality and performance. You may also consider deleting any polys that never face the user when a fixed camera position is utilized. Developing a script in your modeling package should help make quick work of this process. For our “cinema” example case, this process reduced the polycount by 40%.



This image shows the poly optimizations of chairs in the “cinema” mesh. [MODO]

- If you’ve optimized your mesh and still land above the recommended limits, you may be able to push more polys by cutting the mesh into 9 individual meshes arranged in a 3x3 grid. If you can only watch movies from the center grid cell, hopefully some of the meshes for the other grid cells will not draw if they're out of your current view. Mesh visibility is determined by a bounding box and if you can see the bounding box of a mesh, then you're drawing every single one of its triangles. The figure below illustrates where cut points could be placed if the theater mesh requires this optimization. Note that the cut at the back of the theater is right behind the last camera position. This keeps triangles behind this cut point from drawing while the user is looking forward:



[MODO]

- Another optimization that will help rendering performance is polygon draw order. The best way to get your polygons all sorted is to move your theater geometry such that the average of all the camera positions is near 0,0,0. Then utilize the `-sort origin` command-line option of the FBX converter when compiling this mesh (see Step 7 for more information).
- Material : The more materials you apply to your mesh, the more you slow down the renderer. Keep this number low. We apply one material per scene and it's basically one 4k texture (well, it's actually two, because you have to have one for when the lights are on and one for when the lights are off - this is covered in Step 3 below).
- Textures : You'll need to add your two textures to the material that show the theater with the lights on and lights off. The way you do that is you add the texture with the lights on and set its mode to *diffuse color* (in MODO) and set the mode of the texture with the lights off to *luminous color*.

2) Create a screen mesh

- Mesh : Create a quad and have its UVs use the full 0-1 space, or it won't be drawing the whole movie screen.
- Mesh Name : Name the mesh "screen".
- Material : Apply a material to it called "screen". You may add a tiny 2x2 image called "screen.png" to the material, but it is not absolutely necessary.

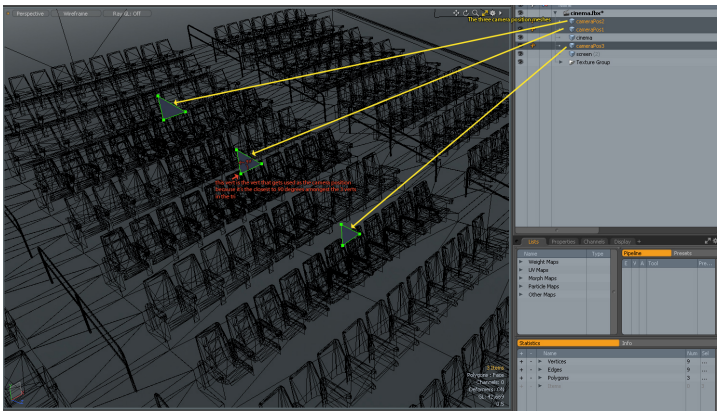
3) Create two theater textures (one with the lights on and one with the lights off)

- Create a CG scene in modo that is lit as if the lights were turned on. This scene is then baked to a texture. Next, turn off all room lights in the modo scene and add one area light coming from the screen. This scene is baked out as the second texture. In the cinema that is included with the SDK, the material is named "cinema." The lights-on image is named "cinema_a.png" and lights-off image is named "cinema_b.png"

Two images for the cinema mesh, with the screen and additive image. [MODO]

4) Create meshes that define the camera/view positions

- A camera/view position for one of the possible seats in the theater is defined by creating a mesh that consists of a single triangle with a 90 degree corner. The vert at the 90 degree corner is used for the camera position. Name the mesh “cameraPos1”. When you process the scene using the FBX converter, use the `-tag` command-line parameter to specify which meshes are used to create the camera positions, and use the `-remove` parameter to remove them from the scene so that they’re not drawn. For example, when converting cinema.fbx, we use `-tag screen cameraPos1 cameraPos2 cameraPos3 -remove cameraPos1 cameraPos2 cameraPos3` on the command line to convert the camera position meshes to tags and remove them.
- Max number of seating positions : The current Cinema application supports up to 8 camera positions.



Three camera position meshes. [MODO]

5) Create meshes that use an additive material (optional, for when the lights are off)

- Different real-world movie theaters leave different lights on while movies are playing. They may dimly light stair lights, walkway lights, or wall lights. To recreate these lights in the virtual theater, bake them to some polys that are drawn additively.
- To make an additive material: simply create a material and append “_additive” to the material name, then add an image and assign it to *luminous color*.

6) Run the FBX converter tool

- To avoid retyping all the FBX converter command-line options, it is common practice to create a batch file that launches the FBX converter. For the example cinema, the batch is placed in the folder above where the FBX file is located:

  \OculusSDK\Mobile\Main\SourceAssets\scenes\cinema\cinema.fbx
  \OculusSDK\Mobile\Main\SourceAssets\scenes\cinema.bat

- The cinema.bat batch file contains the following:

```
FbxConvertx64.exe -o cinema -pack -cinema -stripModoNumbers -rotate 90 -scale 0.01
-flipv -attrib position uv0 -sort origin -tag screen cameraPos1 cameraPos2 cameraPos3
-remove cameraPos1 cameraPos2 cameraPos3 -render cinema\cinema.fbx -raytrace screen
-include cinema\icon.png
```

Note: This is a single line in the batch file that we've wrapped for the sake of clarity.

Here is an explanation of the different command-line options used to compile the cinema.

`FbxConvertx64.exe`	The FBX converter executable.
`-o cinema`	The `-o` option specifies the name of the .ovrscene file.
`-pack`	Makes the FBX converter automatically run the cinema_pack.bat batch file that packages everything into the .ovrscene file.
`-cinema`	The .ovrscene file is automatically loaded into Cinema instead of VrScene when the device is connected.
`-stripMODONumbers`	MODO often appends " {2}" to item names because it does not allow any duplicate names. This option strips those numbers.
`-rotate 90`	Rotates the whole theater 90 degrees about Y because the cinema was built to look down +Z, while Cinema looks down +X.
`-scale 0.01`	Scales the theater by a factor of 100 (when MODO saves an FBX file, it converts meters to centimeters).
`-flipv`	Flips the textures vertically, because they are flipped when they are compressed.
`-attrib position uv0`	Makes the FBX converter remove all vertex attributes except for the 'position' and first texture coordinate.
`-sort origin`	Sorts all triangles front-to-back from 0,0,0 to improve rendering performance.
`-tag screen cameraPos1 cameraPos2 cameraPos3`	Creates tags for the screen and view positions in the theater. These tags are used by Cinema.
`-remove cameraPos1 cameraPos2 cameraPos3`	Keeps the view position meshes from being rendered.
`-render cinema\cinema.fbx`	Specifies the FBX file to compile.
`-raytrace screen`	Allows gaze selection on the theater screen.
`-include cinema\icon.png`	Includes an icon for the theater in the .ovrscene file.

- These are most of the options you'll need for the FbxConvert.exe. Additional options exist, but they are not typically needed to create a theater. For a complete list of options and more details on how to create and convert FBX files, see FbxConvert.txt.
- An example of another command-line option is `-discrete <mesh1> [<mesh2> ...]`. Use this when you cut your mesh into chunks to reduce the number of polys being drawn when any of these meshes are offscreen. By default, the FBX converter optimizes the geometry for rendering by merging the geometry into as few meshes as it can. If you cut the scene up into 9 meshes, the FBX converter will merge them back together again unless you use the `-discrete` command-line option.

Last Update: Nov 10, 2014