12-3-2015

Adapting Architectural Models for Visualization Using Virtual Reality Headsets

Bernardo Martinez
Andrews University, bernardm@andrews.edu

J. N. Andrews Honors Program
Andrews University

HONS 497
Honors Thesis

Adapting Architectural Models for Visualization Using Virtual Reality Headsets

Bernardo Martinez

12/3/2015

Advisor: Dr. Rodney Lee Summerscales

Primary Advisor Signature: [illegible]

Department: Engineering and Computer Science

Adapting Architectural Models For Visualization Using Virtual Reality Headsets

Bernardo Martinez*‡, Dr. Rodney Summercales*, Prof. Ariel Solis†
*Department of Computer Science, Andrews University †Department of Architecture, Andrews University
‡ J. N. Andrews Honors Program

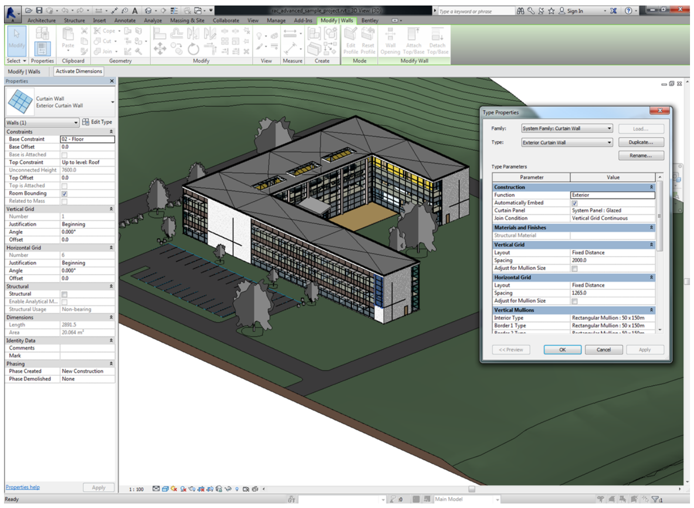
***Abstract*—Business contracts represent a main source of income for Architects. Acquiring these contracts requires the latest and most immersive technology that improves their sales against competitors. Virtual reality provides an in-depth experience that allows clients to have a reasonable assurance that the building meets their physical expectations. Videos and photos are detached and mundane; while they provide some visual representation they will not allow the user to compare his physical characteristics (height, length, width) with a 3D model. In this paper, I describe a procedure for automatically importing 3D models from Revit into Unreal4. I also describe the workflow required which includes constraints and benefits found along the way. As my focus was based on VR, a lot of my work was around the Oculus VR headset which provides an immersive experience into future and current buildings. The software tools I wrote modify 3D models to be compliant with UE4 materials, textures, groupings and it allows developers to split buildings into multiple slices improving performance and polygon counts.**

I. Introduction

Architects need better ways to sell 3D models. Currently they design their buildings and let their clients watch a video rendering of how it will look. Videos and Snapshots do not allow clients to visualize the proportions of walls, windows, doors. Oculus recently developed one of the first practical virtual reality headsets called the Rift. Because the oculus is a new technological gadget, there's a lack of interactions between game companies and architectural models. My task is to create tools and a procedure [1] for architects/artist to utilize Unreal4 [2] and the Oculus Rift to display their 3D models.

A. Related Work

Artists and designers work with tools like 3Dsmax, Maya and Blender to create virtual environments. They usually build their models from blueprints in order to achieve the highest performance and levels of detail required by the industry. Similarly, architects use Revit (see figure 1 to develop blueprints and advertise their buildings, but architects although knowledgeable are restricted by their profession. Recently, there have been only a handful of companies that provide virtual walkthroughs. However, their software and methods are proprietary and not publicly documented. ArchVirtual and Virtualdutchmen are small companies that develop walkthroughs with the Oculus Rift Dk2 but their targets are corporate environments. For my project I am targeting the small to medium markets. IrisVr relates to my project, but they focuse on a bigger market due to their multiple angel investors. Architecture models are mostly visualized through software tools like Lumion, which allow architects with small budgets to create videos of their 3D Models. Lumion has certain limitations that Unreal4 can fix for example: Lumion is a rendering engine, it wont allow the world to be animated, since everything around you is just a photo map. The inherent limitations of Lumion are only offset by it's plugins.



Constraints	
Base Constraint	02 - Floor
Base Offset	0.0
Base is Attached	☐
Top Constraint	Up to level: Roof
Unconnected Height	7600.0
Top Offset	0.0
Top is Attached	☐
Room Bounding	✓
Related to Mass	☐
Vertical Grid	
Number	1
Justification	Beginning
Angle	0.000°
Offset	0.0
Horizontal Grid	
Number	6
Justification	Beginning
Angle	0.000°
Offset	0.0
Structural	
Structural	☐
Enable Analytical M...	☐
Structural Usage	Non-bearing
Dimensions	
Length	2891.5
Area	20.064 m²
Identity Data	
Comments	
Mark	
Phasing	
Phase Created	New Construction
Phase Demolished	None

Type Properties

Family: System Family: Curtain Wall
Type: Exterior Curtain Wall

Type Parameters

Parameter	Value
Construction	
Function	Exterior
Automatically Embed	✓
Curtain Panel	System Panel : Glazed
Join Condition	Vertical Grid Continuous
Materials and Finishes	
Structural Material	
Vertical Grid	
Layout	Fixed Distance
Spacing	2000.0
Adjust for Mullion Size	☐
Horizontal Grid	
Layout	Fixed Distance
Spacing	1265.0
Adjust for Mullion Size	☐
Vertical Mullions	
Interior Type	Rectangular Mullion : 50 x 150m
Border 1 Type	Rectangular Mullion : 50 x 150m

Fig. 1: AutoDesk Revit workflow environment

One of the main reasons artists and architects will not use Revit models is due to the high polygon count that certain blueprints are based of. Similarly, exporting a file with the FBX format would destroy the object layers, requiring an artist to join each polygon face manually making it an exhaustive task. Revit provides a uniform editor that allows architects to test their models under different views (2D and 3D). Sadly, Revit was made for precision and easy/quick development, regardless of optimization. There are certain tools that crush polygons and reduce the size of a mesh, but they destroy the appearance of the object.

In order to incorporate Virtual Reality I used the Oculus Rift, a wearable device that allows the user to perceive its environment with an extra axis of depth. Currently this device has been used by video games and simulations to demonstrate a fully fledge experience on a virtual environment. Although there are alternative devices to the Oculus, its market provided the biggest pool of potential clients.

II. Methodology

By following my procedure and using the automated tools I wrote, architects and artists can regroup objects into layers, replace proprietary materials and split a building into multiple files. All the models are compatible with the Oculus Rift and optimized to improve frame rate. Given a FBXmodel from Revit, the user imports the model into 3Dsmax, overrides the proprietary materials, with standard materials (ex. phong) and run the C++ automated scripts. Once the scripts run the user will find their original model split, and reorganized by layers.

As my research developed I encountered constraints that played an important role. Some of them related to material incompatibilities, Polygons not optimized [3], shadows and many others. Figure 7 showcases each polygon as a black line, notice multiple lines at vertices and edges. I developed solutions to address each one of them and I grouped the solutions into tools that allow a significant decrease on development time. I divided my time into three phases which included multiple iterations of sprints. Each sprint consistent of research, development and testing of prominent solutions. Phase one was based on familiarizing with the environment, learning the tools(3Dsmax,UE4,Revit) [4], [5], [6] and figuring which aspects could be optimized. During phase one I programmed on 3Dsmax scripts and developed the grouping and repainting algorithms. Phase two consisted on improving and enhancing the capabilities of some scripts while researching how to improve speed and performance. During this faced I looked at the FBX SDK (software development kit), which allow me to get ride of 3Dsmax interface for the most part. By avoiding the need of loading 3Dsmax into memory times were significantly reduced. Phase three consisted on expanding my research through an algorithm that would split buildings into separate pieces that could be loaded into memory independently. As an example see Figure 5.

I summarized results and optimize the scripts, allowing clients to have an easier experience. Details about specific sections will allow a deeper understanding of the requirements and constraints of building such an applications.

A. *Analyzing 3D models*

Initially, I tested 3D model's incompatibilities (parameters that will make significant difference between buildings). I loaded different models with different characteristics to identify the interactions between the model and 3Dsmax. Subsequently, I explored different procedures to import Revit files into UE4[7]. Different solutions drove slightly different outputs. At first, my scripts on 3Dsmax would allowed me to replace encrypted material but the editor will only run if I opened 3Dsmax interface and loaded the model. Due to the increasing size of my 3D models, the interface become an issue; I had extra overhead of loading the whole interface. There are two different ways of importing objects from Revit, either you import the rvt file (this is also referred as a link to Revit) from 3Dsmax, or you export Revit as a FBX and import it into 3Dsmax. Both ways have some advantages and disadvantages. For example: I generated the FBX from 3Dsmax using a link to Revit rvt file. Once imported, the mesh

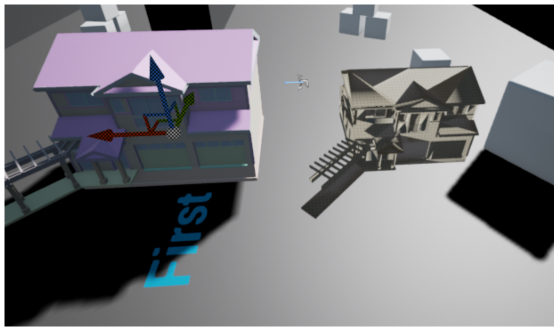


Fig. 2: Light-map wraps around the image

consisted of only one color. A single material meant that the level of detail was crushed(level of details refers to the number of colors inside an object). Figure 2 contains an example of an object imported with a single material and another with material instances being remapped. Similarly I tried exporting an FBX file from Revit and importing it on 3Dsmax, but then I would get hundreds of different pieces (every object in a layer would be broken into its own instance) that Unreal4 welded (joined together) in order to save space. Welded polygons would only take one material. After researching why material detection was not being applied inside Unreal4, I found that Unreal4 only supported certain kinds of materials, and that in order for this feature to work I had to use either mental rays or Vrays. Mental rays and Vrays are production quality rendering applications that have their unique set of materials. Fortunately Mental rays were provided already by 3Dsmax. Once I applied the right materials, and group the objects inside the mesh into 64 (threshold of objects per mesh in UE4) or less layers, I acquire the ability to modify textures and colors at will on the surfaces that were needed.

B. *Multimaterial vs Single Material*

Materials imported from 3Dsmax had to re-mapped into standard phong/Mental Ray materials [8]. Unreal4 does not support V-Rays hence one of my scripts had to deal with remapping and reducing the amount of materials in a object. Importing directly from Autodesk Revit was also tested, but their materials are encrypted and they do not allow any other engines to use them. Interestingly enough, textures[9] that were not attached to a specific material transfer in perfectly. Figure 4 represents the same 3Dmodel with different levels of detail.

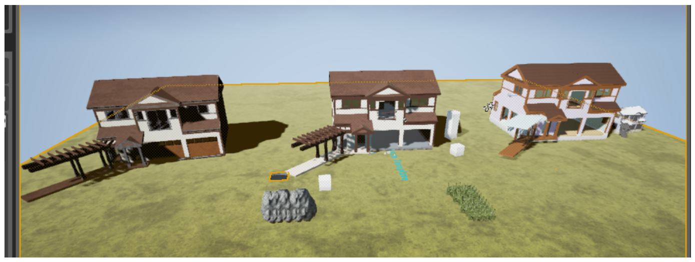


Fig. 4: Three different models, each one made with a higher material count.

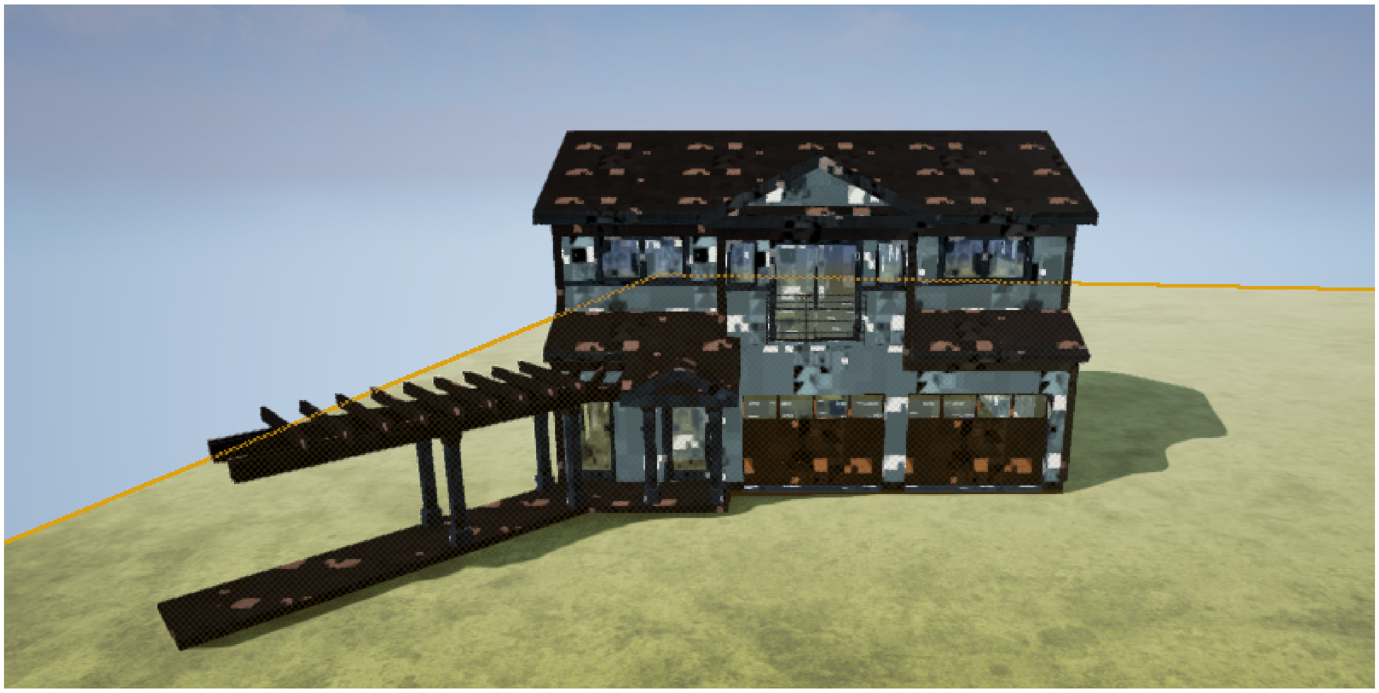


Fig. 3: Static Light-map wraps around the image

C. Static & Dynamic Shadows

Shadows played a major role in how realistic an object was perceived. Unreal4 allowed shadows to be calculated dynamically or statically with Lightmaps. When I started the project, Unreal4.3 did not have a good dynamic lighting implementation, so I had to turn to static lights. Static lights are harder than normal due to the high level of detail inside architecture models (too many polygon faces)[10], [11]. Traditionally, artists create models that are as low poly (polygon count) as possible allowing them to modify polygon faces manually into a 0 to 1 map. Because my model was not optimized, I would get as much as 10 times more polygon faces that an artist would get. Figure 6 provides an example on how static lights affect a 3D model. Static lights require handpick lightmaps in order to achieve inside and outside corners that reflect realistic details. To illustrate how handpicked shadows can improve or destroy a model looked at Figure 3. Although some algorithms exist the are only useful for meshes are optimized below a threshold.

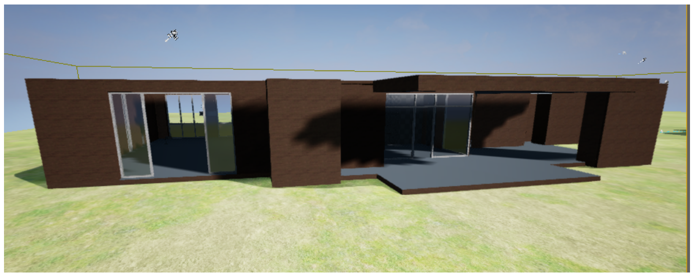


Fig. 6: Static LightMap dims real time lightening.

Since material conflicts had a higher priority I decided to work on shadows later, since later updates might help resolve this issue. About 3 months later, Unreal4 had an improved algorithm in which dynamic lights increased their quality about 75%, so I decided to apply them. By the end of the first semester, I developed scripts that would replace the materials and produce a mesh I could play with on Unreal4. Throughout the second semester I increased the quality of my scripts as well as the quality of my demos. By Spring Break, I realized that I needed to improve my performance, so I looked into optimizing the process, which brought me over to the FBX SDK in C++. I had seen the SDK before, but it lacked documentation and it was at this point that I felt confident of tackling this task. Within the FBX SDK I was able to enhance the speed of my procedure, since I did not have to deal with the graphical interface of 3Dsmax.

D. Client Optimizations

Since my project was focused on getting architects as prospective clients, I had requests to increase the quality (materials, lightening, textures) of the final product which brought me onto designing a workflow for optimizing and increasing the appeal of the models inside UE4. Three sets of models were made utilizing different technologies that allowed Prof. Ariel Solis and I to showcase to one of his clients the potential of UE4. My first design used simple materials and one or two positional lights appealing to the afternoon look inside the house. At this point, materials consisted only of simple colors and standard lights. The second design incorporated new assets that I brought and transferred over from Unreal4 demos and their market place. Besides including complex objects, this design allowed us to explore textures, which enhanced the level of detail in the mesh. As the quality increased some

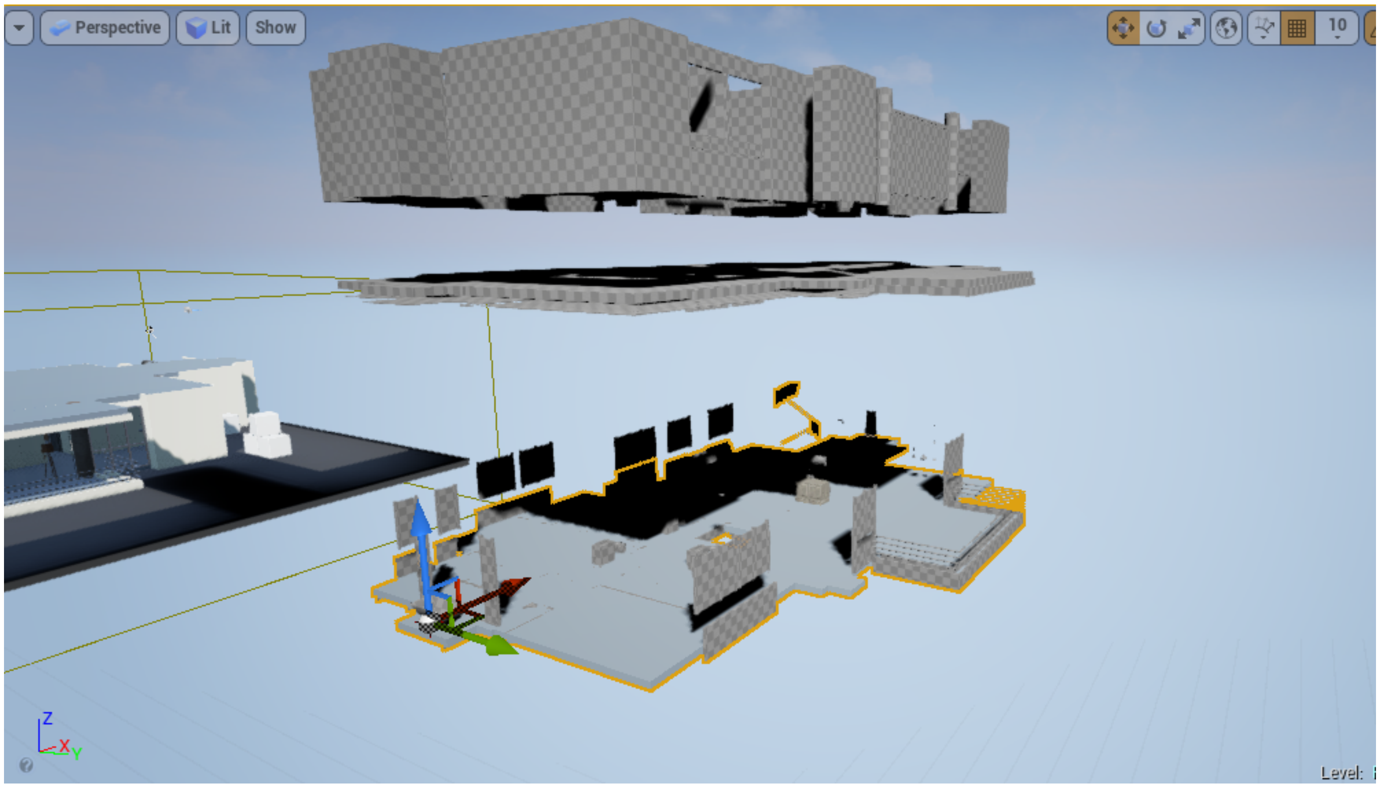


Fig. 5: Optimization algorithm Output. Layers developed horizontally allow faster loads time

issues related to the geometrical positioning on Revit also came to surface. We encountered some glitches in the level due to walls and columns overlapping. As objects with different materials overlap Unreal4 will flicker, due to UE4 double buffer real time rendering. The final design was made with some alterations from the main script that allowed the user to have a high level of accuracy when deciding which items to pick and join as a layer for later usage in the project. By eliminating most glitches the third demo became the best of on performance and appeal.

E. Scaling inside UE4

3Dsmax has different measures when exporting onto UE4 due to the UE4 grid set in cm, while 3Dsmax was in inches. Although both could be set to the same scale, their standards were different.

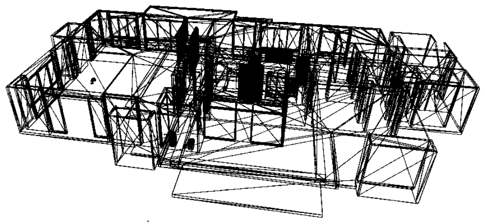


Fig. 7: Multi-polygon object

F. UE4 updates

UE4 is a professional game engine. Periodic updates to the engine required extra time to adapt my procedures to the newer version of the engine.

G. UE4 level streaming

While working on this project I got to test three different structures which were a small house design, a small one-floor apartment, and an apartment complex. Each model increased in size, which became proportional to the loading time inside Unreal4. This load dropped the FPS (Frames per Second) from a good 35 to 3-6 frame. Lag was present at that rate mainly because of the number of objects loaded into memory at one time. To increase performance I researched level streaming. It allow different parts of the world to be loaded on demand, saving resources that are highly needed for the oculus rift to perform in optimal conditions. Level streaming works best if objects can be split into units. That is why I turn into optimizing the buildings by breaking them into different levels.

H. FBX SDK

By creating an optimized interface on C++ with the FBX SDK [12], the amount of time parsing a file was significantly reduced. This allowed the file to be loaded on laptops with a smaller graphics card. The materials remapped and automatic layers were done in both in 3Dsmax scripting language and C++. After I translated my code from 3Dsmax scripting language to C++ I incorporated new features. I developed an algorithm that will analyze a 3D mesh and depending on the global coordinates of each object, it will place them into a new 3D model and export them. Once a 3D model is loaded into 3Dsmax, there are two parameters that had to be changed before my scripts can split a mesh. First the mesh coordinate system has to be changed onto a global system. 3D models can have multiple coordinates systems per object. Secondly, the encrypted materials have to be replace by any standard material. Subsequently, I will run my script which parses all the nodes inside the FBX mesh placing the ones below a threshold inside different buckets and then exporting the buckets as different files. In order to have a smart slicing algorithm, I read the size of the mesh and split it into the amount of pieces the user needs. For objects which go above a certain threshold, the algorithm will place them onto the lower bound bucket. Through my testing CPU times were reduced by a third. A high-performance graphics computer that took 11.3 minutes to load a model was cut down to 3.8.

Algorithm 1 My algorithm

```
1: function SplitMesh( SceneRef, FBXNode, index)
2:     $NodeCounter \leftarrow$ number of Nodes per $Mesh$
3: $top$:
4:     if $NodeCounter == 0$ then return false
5:     $zvalueofmesh = FBXNode.zaxis()$
6:     $arrayofboundaries$                                  $\leftarrow$
   Global variable previously initialized
7: $loop$:
8:     if $index == 0$ then
9:         if $zvalueofmesh < arrayofboundaries[0]$ then
10:            continue
11:        else
12:            $SceneRef.AddChild()$                        $\leftarrow$
   Transfers node onto a new scene
13:    if $index > 0$ then
14:        if $FBXNode.GetChild! = null$ then
15:            $zvalueofmesh$                               $\leftarrow$
   Updates current node on $Mesh$
16:            if $izvalueofmesh > arrayofboundaries.lower$ AND
   $izvalueofmesh < arrayofboundaries.higher$ then
17:            $SceneRef.AddChild()$                        $\leftarrow$
   Transfers node onto a new scene
18:        else
19:            continue
20:        goto $loop$.
21:        close;
```

III. Results

Multiple demos were made to showcase the different effects that materials, textures, lightening, and others have on objects made with Revit. Revit makes 3D objects harder for artists and programmers but it has a lot powerful tools for sketching. The splitting algorithm allows architects to load big buildings on a fraction of the time, by following the procedure I developed they inherent a significant time reduction on research and development.

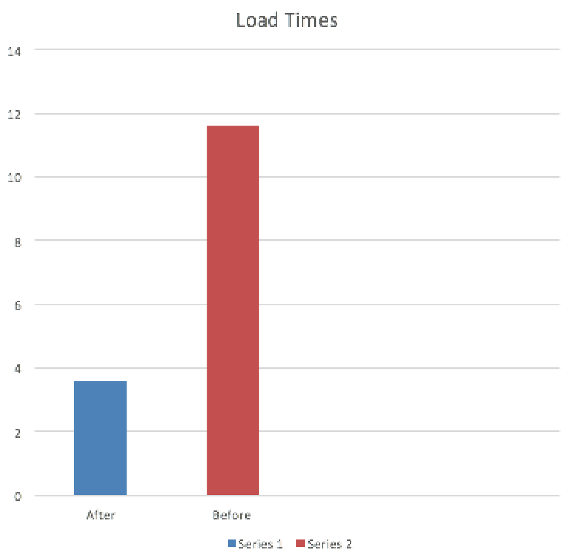


Fig. 8: Performance difference from an object that is split.

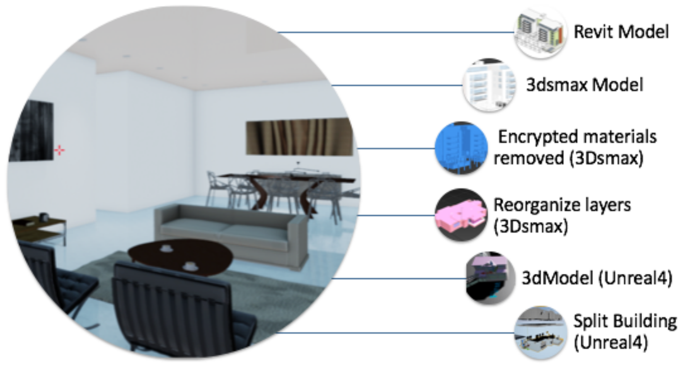


Fig. 9: Workflow developed, from Revit to Unreal4

IV. Conclusion

Artists might still need training to use some of my scripts, but once they get it they will find it incredible time saving. As my future work I would like to work along with an artist. I focused on getting the algorithms and the appearance, which limited my time to focused on either one.

V. Acknowledgement

I want to thank Prof. Ariel Solis from the architecture department for providing the 3Dmodels and some inside knowledge on Revit Architecture. He was key on the success of this project due to his knowledge and expertise. To my honors advisor Dr. Rodney Summercales, I am grateful for your assistance and suggestions throughout my project.

Appendix

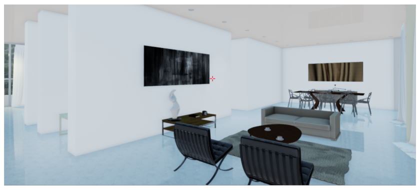


Fig. 10: Frontal Image of the living room with multiple objects

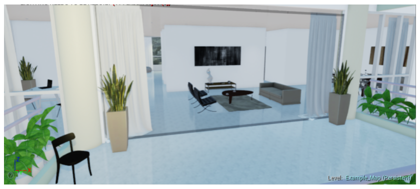


Fig. 11: Final Demo, showcasing a side view of the appartment

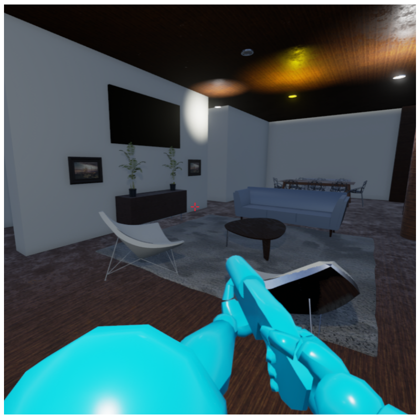
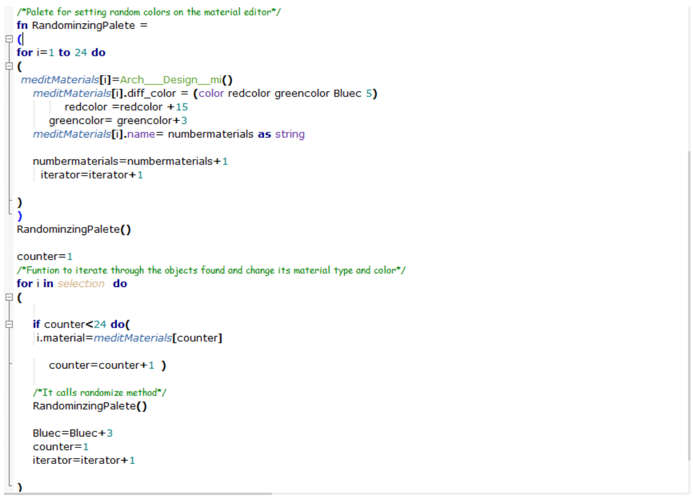
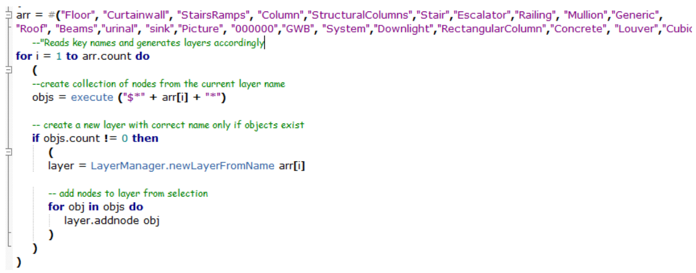


Fig. 12: Evening Demo with multiple coloring on the roof



```
/*Palete for setting random colors on the material editor*/
fn RandominzingPalete =
(
for i=1 to 24 do
(
 meditMaterials[i]=Arch___Design__mi()
   meditMaterials[i].diff_color = (color redcolor greencolor Bluec 5)
        redcolor =redcolor +15
     greencolor= greencolor+3
   meditMaterials[i].name= numbermaterials as string

   numbermaterials=numbermaterials+1
    iterator=iterator+1

)
)
RandominzingPalete()

counter=1
/*Funtion to iterate through the objects found and change its material type and color*/
for i in selection  do
(

   if counter<24 do(
    i.material=meditMaterials[counter]

     counter=counter+1  )

   /*It calls randomize method*/
   RandominzingPalete()

   Bluec=Bluec+3
   counter=1
   iterator=iterator+1

)
```

Fig. 13: 3Dsmax Scripting Language, Randomizing materials



```
arr = #("Floor", "Curtainwall", "StairsRamps", "Column","StructuralColumns","Stair","Escalator","Railing", "Mullion","Generic",
"Roof", "Beams","urinal", "sink","Picture", "000000","GWB", "System","Downlight","RectangularColumn","Concrete", "Louver","Cubic
   --"Reads key names and generates layers accordingly
for i = 1 to arr.count do
   (
   --create collection of nodes from the current layer name
   objs = execute ("$*" + arr[i] + "*")

   -- create a new layer with correct name only if objects exist
   if objs.count != 0 then
      (
      layer = LayerManager.newLayerFromName arr[i]

      -- add nodes to layer from selection
      for obj in objs do
         layer.addnode obj
      )
   )
)
```

Fig. 14: 3Dsmax Scripting Language, Regrouping objects

References

[1] (2015) Game asset production pipeline. [Online]. Available: http://www.digitaltutors.com/tutorial/1635-Game-Asset-Production-Pipeline-in-Unreal-Engine#play-42004

[2] EpicGames. (2014) Runnning unreal engine. [Online]. Available: https://docs.unrealengine.com/latest/INT/GettingStarted/RunningUnrealEngine/index.html

[3] Polygon crusher for 3dsmax. [Online]. Available: www.moontools.com

[4] (2012) Fbx sdk documentation 2013. [Online]. Available: http://docs.autodesk.com/FBX/2013/ENU/FBX-SDK-Documentation/index.html

[5] Sanvfx, "3ds max — tutorials."

[6] EpicGames. (2014) Unreal4 engine documentation. [Online]. Available: https://docs.unrealengine.com/latest/INT/

[7] (2014) Intro to bim modeling. [Online]. Available: https://www.youtube.com/channel/UC0y73dD7p4gjV2x9etIeL4w

[8] S. Butler. (2009) Random material to random object script. [Online]. Available: http://www.scriptspot.com/forums/3ds-max/scripts-wanted/random-material-to-random-object-script

[9] J. E. Ltd. Remove textures in 3dsmax. [Online]. Available: http://www.polycount.com/forum/showthread.php?t=58410

[10] (2014) Invalid lightmap settings. [Online]. Available: https://forums.unrealengine.com/showthread.php?888-Invalid-lightmap-settings

[11] B. S. (2010) Creating lightmaps in 3ds max 2009. [Online]. Available: http://www.dxstudio.com/guide_content.aspx?id=0a927204-bb25-4b4f-8a71-b744c08cb8de

[12] (2015) Autodesk fbx sdk 2015. [Online]. Available: http://help.autodesk.com/view/FBX/2015/ENU/?guid=__files_GUID_75CD0DC4_05C8_4497_AC6E_EA11406EAE26_htm

[13] Twinsavior. (2011) Autodesk revit architecture section box. [Online]. Available: https://www.youtube.com/watch?v=OuM3aacc-aQ

[14] j0k. (2011) How to reduce the number of polygons of 3d model using 3ds max 9.0. [Online]. Available: http://stackoverflow.com/questions/5217894/how-to-reduce-the-no-of-polygons-of-3d-model-using-3ds-max-9-0

[15] AutoDesk, "How to -introduction to the tutorials," 2015. [Online]. Available: http://docs.autodesk.com/3DSMAX/15/ENU/MAXScript-Help/index.html?url=files/GUID-676FB825-84C1-4708-A398-993266E4D2AD.htm,topicNumber=d30e98789

[16] C. Grant. (2014) 3ds max — tutorials. [Online]. Available: http://www.scruptspot.com/3ds-max/tutorials

[17] O. VR. (2015) Start building. [Online]. Available: https://developer.oculus.com

[18] EpicGames. (2015) News epic games. [Online]. Available: https://www.unrealengine.com/news/