

# Visualization Insider

## Managing System Resources



## Introduction

A frustrating reality of working in any 3D program is waiting on the computer to process information. In the ideal 3D world, you could build all your scenes without regard to file size, available RAM, CPU speed, rendering times, or any number of things that complicate the design process. However, since most of us have to be aware of our computers' limitations, we have to manage the way we build objects and assemble our scenes. In order to work efficiently, we should do whatever we reasonably can to minimize our wait.

At 3DAS, we're often asked what kind of hardware we use to create and render our scenes. Students are often surprised to hear that for many years now we've relied primarily on average speed computers. At the time this article was written, the typical computer in our office is a Core 2 Duo Dell laptop with 2GB RAM. Many new users entering into the field of architectural visualization have misconceptions that producing good work requires top end hardware, and while may have been true 15 years ago, today it's simply not the case.

There are several different situations in which the 3ds Max user is forced to wait on the computer. Three of these, in particular, are situations in which good procedures and practices can spare you from long and unnecessary waiting; transferring files, refreshing viewports, and rendering

Most other times, 3ds Max is waiting on the user for input, but for each of these you are forced to spend at least some time waiting on 3ds Max and your computer, depending largely on how you manage your scenes. So what are the attributes of a scene that affect each of these three situations, and what can you do to minimize your wait during each of these processes? In this article, we're going to take a close look at some tools and procedures that can improve system resource management so that you don't have to worry about running out of RAM or having Max crash when you render very large scenes. As a supplement to this article, check out the sample chapter, Chapter 17 - Scene Assembly, from [3ds Max 8 Architectural Visualization](#), available through week 12 of the [Visualization Insider](#). In that chapter, you'll find several ways to minimize the time you wait on your computer during each of the 3 situations listed above.

## Handling Individual Objects

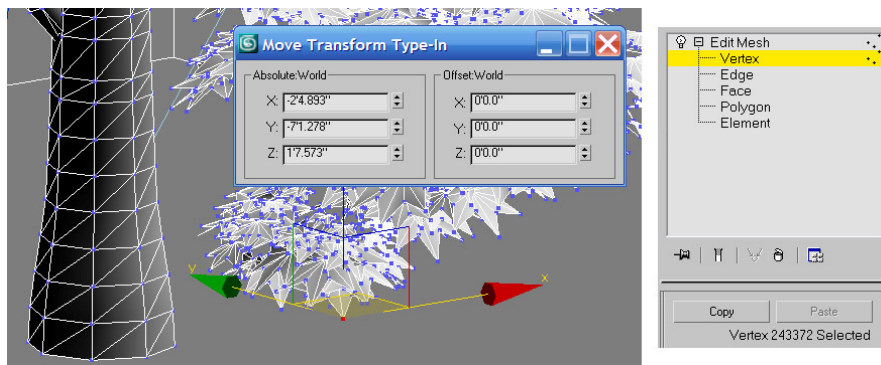
Let's jump right into a scene that demonstrates the importance of how to handle individual objects. In the image below, we have a single tree by itself which was created with the plugin called Tree Storm by Onyx Computing. This is a great plugin I use on virtually all exterior scenes to quickly and easily populate scenes with realistic 3D trees. If you have the Tree Storm plugin and want to follow along using this particular object, open the file called **aesculus.max** (courtesy of Onyx Computing). If you don't have Tree Storm, in just a moment I'll tell you how you can follow along with a different file after these first few steps of this demonstration are complete.



The very first thing I want to do is keep track of my object and scene face counts, so I'll press the keyboard shortcut **7**. When I do, I'm shown the scene face count and vertex count in the active viewport. I can also go to the viewport **Configuration** menu, and in the Statics tab I can enable the **Total+Selection** option which allows me to also see the vertex and face count of individual objects when I press the shortcut **7**.

This tree is parametric based, which means that its structure, i.e., all of the branches and leaves etc, is controlled and generated through parameters. So Max doesn't have to store the information for each individual leaf on this tree because the size, shape, and orientation of the leaves are all derived from algorithms. This same concept applies to a primitive object like the teapot. If I want to change the smoothness of the teapot by changing the number of segments, of course all I need to do is increase the segments parameter.

So with just this one tree the face count is 267,000 faces...quite large for an individual object and quite large even for a tree. But if I save this file with the tree in parametric form, the file size will only be about 360KB. Not large at all, especially when you consider that an empty scene is going to generate a file size of about 220KB anyway. This type of object is completely different from the Edit Mesh or Edit Poly objects, whose structure and display are based on subobject components like vertices, faces, etc. If I were to collapse the tree in this scene to an editable mesh, then the tree is no longer parametric based and a lot more information would need to be stored for this object. In fact, all of the tree's vertices need to have a position, rotation, and scale value defined, and this means a much larger file size is generated.



If I collapse this tree with 267,000 faces, the file size increases to 22MB because Max can no longer define the tree's structure by algorithms or formulas. Instead, it must store the X, Y, and Z value of every individual vertex for each of the 3 transform types; position, rotation, and scale. For an object with 267,000 faces, it shouldn't be hard to see why the file size is now 21MB.

Now to demonstrate another quality of how information for objects is stored, I'll create a teapot with 64 segments. With 64 segments, the teapot's face count is 262,000...almost the same as the tree. But if I collapse this teapot, notice in the image below that the file size is only 11MB, which is half of what the tree's file size is. The reason the two file sizes are so different even though they contain the same number of faces is that the teapot contains half as many vertices. And the reason it only needs half as many vertices as the tree with the same number of faces is because all of the faces in the teapot share vertices with adjacent faces. This doesn't mean that the rendering times will be so vastly different because rendering times are determined more by face count. So the vertex count drives a scene's file size and face count drives render times.

h:\My Documents\3dsmax\scenes		
Name	Size	
aesculus	364 KB	
aesculus-collapsed	21,684 KB	
teapot	11,672 KB	

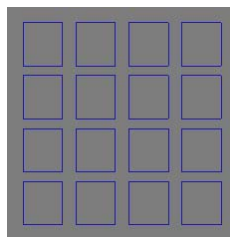
Now collapsing this tree to an editable mesh or poly is not the same as adding the Edit Mesh or Edit Poly modifier to the base object. If you simply add one of these modifiers, you haven't really added any information because until you start making changes to sub-objects within the added modifier, the vertices won't require position, rotation, or scale information...they will still be stored through the base parametric object. Therefore, the file size shouldn't change, as shown below with the file named **aesculus\_with\_1\_modifier\_unchanged.max**. But if you do go into subobject mode and move all the vertices from their initial location, once you save the file you are saving new position information for all the vertices and the file size becomes much larger. This can be seen in the file named **aesculus\_with\_1\_modifier\_changed.max**. If you add another Edit Mesh modifier, and repeat the same process of transforming vertices, or any other subobject, you will continue adding information to the file, also shown below.

Name	Size
aesculus	352 KB
aesculus_with_1_modifier_unchanged	352 KB
aesculus_with_1_modifier_changed	4,760 KB
aesculus_with_2_modifiers_changed	9,168 KB
aesculus_with_3_modifiers_changed	13,576 KB
aesculus-collapsed	21,684 KB
teapot	11,672 KB

Whether or not you have the Tree Storm plugin, you can follow along from this point on with the file named **aesculus-collapsed.max**. This is simply the tree in collapsed form.

Now before I go any farther I want to open the Windows Task Manager and keep this open for the remainder of this discussion on system resource management. Within the Windows Task Manager there are several different tabs, one of which displays 3 very important values that you should always keep an eye on when working in large, resource hungry scenes; they are the processor usage, the page file and the available RAM.

Now for the next demonstration, I need to make 15 duplicates of this collapsed tree and place them in a 4 x 4 grid. Since this tree has so many faces, displaying it in its entirety is a strain on the graphics card, so what I should do before making the duplicates is to turn this tree into a box. Working in the viewports will be much easier once I do this. When I make the duplicates (as shown below), I want to make sure that they are created as instances rather than copies so that they don't increase the file size. The image below shows the 16 instanced trees as boxes.



If you select one individual tree, you'll notice that the edit mesh modifier is shown in bold, indicating it's an instance of another object. Well now I want to select all of the trees and collapse them all into editable meshes. But since each of these trees is going to be a very processor and memory hungry operation, I want to pay close attention to the Windows Task Manager, specifically the available memory. As soon as the collapse is executed, the available RAM drops by about 400MB.

Before Collapsing		After Collapsing	
Physical Memory (K)		Physical Memory (K)	
Total	2095480	Total	2095480
Available	1070984	Available	641960
System Cache	812892	System Cache	646728

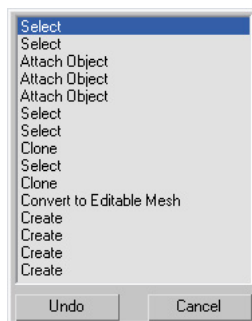
I'll save this file again and give it the name **16-trees-collapsed.max**, and when I do I notice another drop in RAM, this time about 200MB. This is only temporary and the RAM returns to what it was before the save, but it's important to realize that with large files, the simple act of saving a scene can cause your computer to run out of memory and crash. Also notice that the file size has ballooned to over 340MB, which not surprisingly is 16 times the file size of one collapsed tree.

Before I render this out, I want to demonstrate one more thing that concerns RAM. What I want to do is attach a few of the trees together and see how the available RAM is affected. So with one tree highlighted, I'll go to the Edit Mesh modifier in the Command panel and using the Attach command in the Edit Mesh modifier I'll attach one of trees from the Select Objects dialog box. Before I complete the 1<sup>st</sup> attachment, I want to check my available RAM. Before the 1<sup>st</sup> attachment, I have 865MB available, as shown below, and after attaching the 1<sup>st</sup> tree the available RAM drops 116MB. After a 2<sup>nd</sup> attachment, another 143MB is consumed and by the 4<sup>th</sup> attachment I'm down to 193MB and probably within one attachment of crashing Max. Now the attach command that I just executed was extremely memory hungry because essentially, Max has to store the original version of all these trees as individual objects and an additional version with some of the objects attached together. Needless to say, more complex operations usually require more memory and they can often crash Max if you don't pay attention to the available RAM.

Physical Memory (K)		Physical Memory (K)		Physical Memory (K)		Physical Memory (K)		Physical Memory (K)	
Total	2095480	Total	2095480	Total	2095480	Total	2095480	Total	2095480
Available	865476	Available	749208	Available	606188	Available	421804	Available	193376
System Cache	845176	System Cache	737652	System Cache	631372	System Cache	535508	System Cache	305808

Before 1st attachment      After 1st attachment      After 2nd attachment      After 3rd attachment      After 4th attachment

If I were to start rendering these 16 collapsed trees right now, I would really be doing myself a disservice because I am giving Max less memory to work with and it would almost certainly crash during the render process. I should instead free up this memory that was consumed by the attach commands and then render the scene. If I right-click the undo icon, I'm presented with the last 20 steps that I conducted in Max, as shown below, and of course, I can go back to an earlier step before the trees were attached together. But this won't free up any memory because the attach command that I executed would only be moved to the redo list. To free up this memory, I could reset Max and load the scene again, but since the scene file size is so large, this will take longer than I want to wait. So instead, all I need to do is create an object and change a parameter several times until the memory heavy attach commands are purged from the undo list.



So I'll create a sphere and change the radius several times and as I change it, I see in the undo drop down list, that the attach commands keep getting pushed farther down the list. After 20 new commands are performed, the available RAM is returned close to what it was before the trees were attached. This was just a demonstration to point out why you should always keep a close watch on your system resources and why you should always clean your RAM prior to render.

Having covered a few fundamental concepts about how Max stores information, it's time to look at all the different ways objects can be handled and the reasons why different objects should be

treated differently. Many users simply create the objects they need, apply the necessary modifiers, and give little thought to what those objects do to their workflow and efficiency. For small scenes, you might be able to get away with working this way, but as scenes grow in size, so does the need to consider the following questions for each individual object.

- Should the object be collapsed or left as is?
- Should the object exist as an instance, Xref or neither?
- Should the object be attached to other objects or be left as is?

Let's look at each of these questions closely and the factors that should guide your decision on how to answer each of these questions for each type of object.

## Should the object be collapsed or left as is?

If you collapse an object to an editable mesh or editable poly, you lose all modifiers listed in the modifier stack. The obvious downside to this is that you lose the ability to change parameters within any of the modifiers in the modifier stack. If you know that making further modifications to the object is unlikely or if you decide that you can simply retrieve an original uncollapsed version of the object saved in another file, then collapsing the object isn't a problem. Another downside is that you may significantly increase the scene file size the same way the parametric tree increased the file size when it was collapsed. For some modifiers this won't be an issue, but for others, such as Mesh Smooth or Turbo Smooth, collapsing the object with these modifiers means the object will have a far greater number of vertices to account for individually rather than through algorithms.

One not so obvious advantage to collapsing objects is the reduced time it takes to load the scene file. When Max loads a file, every object has to be rebuilt with the modifiers applied to them meaning that if you apply a modifier to an object and that modifier takes 10 seconds to be computed, that object alone will increase the file load time by ten seconds because it has to be reapplied during the loading process. To see this clearly, reset Max, create a simple teapot with the default 4 segments, add the Tessellate modifier and change the Iterations to 4. Notice that when you change the iterations to 4, it takes about 10 seconds to tessellate the teapot. Now save the file and reload it. When you do, you should see that it takes about 10 seconds longer to load the file than it would to load an empty scene. With 4 iterations in the tessellate modifier, the teapot should contain just over 250,000 faces. If you collapse the teapot now and save the file, you'll have a much larger file size but the file will open in just a second or two. So clearly, some thought should go into how to handle certain objects in your scene, especially the most complex, high-poly objects.

Another type of object that can significantly increase the time it takes to load a file is the compound object. This type of object includes Booleans, scatters, shapemerges, etc. Whenever you create a compound object, the original objects are still stored in the file's memory and can be retrieved at any time. What you perhaps didn't know is that every time you load a file with a compound object, that object gets recreated just as an object with a modifier does. Many compound objects can take a good deal of time to calculate, so if your scene contains a large number of them, then be warned you're scene may take a long time to load. In most cases, it's better to collapse the compound object and incur and slightly larger file size than it is to leave the compound object as is and incur a longer load time. So if a scene takes an excessively long time to load, check for these types of situations.

## Should the object exist as an instance, Xref, or neither?

The more complex an object, the more likely it becomes a candidate for an instance or an Xref. If you have 2 complex, high-poly objects, such as 2 of the same kind of tree, it may be desirable to keep them unique so that one can be modified without affecting the other. But remember that file sizes can quickly get out of control when duplicating high-poly objects as copies rather than instances. With Xrefs, you don't have to worry so much about the main scene file size getting out of control, but you will likely run into equally challenging problems, such as longer load times and more difficulty in sharing files over a network.

If you're a V-Ray user, you can (and almost certainly should) use the V-Ray proxy feature on high-poly objects. With this feature, you can literally render a forest of trees like the one shown at the beginning of this article. You can render hundreds of these trees, consisting of tens of millions of polygons, without having to worry about excessive file sizes and load times, or running out of RAM. For more information about the use of V-Ray proxies, check the future article for week 25 of the [Visualization Insider](#) entitled, [An In-Depth Look at the V-Ray Proxy](#).

In Chapter 17 of [3ds Max Architectural Visualization](#), several different scenarios are discussed for when you might choose to make instances rather than copies and vice versa, but one disadvantage to creating instances is that you can't attach instanced objects with any other object. You can certainly group instanced objects together which can help in object management, but you cannot attach instanced objects. Since we're on the subject, let's now discuss the benefits of attaching objects together.

## Should the object be attached to other objects or left as is?

There are two important benefits of attaching objects together. First, attaching one object to another means that instead of having to manage two different objects, you will only have to manage one. That can make things like applying materials and mapping much easier and it can also facilitate selecting and displaying objects. This is especially true when multiple users work on the same project. Few things are as aggravating in 3D as opening the Select Objects dialog box only to see thousands of objects, most of which are labeled as unintelligently as Line01, Box03, Rectang32, etc.

A second advantage is that you reduce the total scene object count which can help reduce rendering times as well as the total RAM consumed. For large scenes with many objects, you may notice that the very first message that appears when you render says 'Preparing Objects'. You might not see this for small scenes if the message only lasts for a fraction of a second but when a scene contains a large number of objects, depending on how the objects were created, the Preparing Objects phase of rendering can take several minutes to process. This is simply an unnecessary aggravation that can be avoided by attaching similar objects with similar materials together. Grouping objects does not change the number of objects in a scene, so other than facilitating management of objects, grouping objects has no bearing on bearing on workflow.

## Summary

This chapter has shown why it's so important to know how to build and assemble a scene without excessive file sizes or face counts. Someday, we may no longer have to be concerned with such issues, but for the foreseeable future we should always try to use the tools available to allow for efficient work and prevent wasted time. I suspect that no matter how fast computers become, software developers will always fill your plates with more and more processor-hungry and memory-consuming programs and resources. If this is true, then the topics covered in this article will always apply.