Karl Lewis
Real Time Debugging

Project 4: The Bigger Badder Bug Report

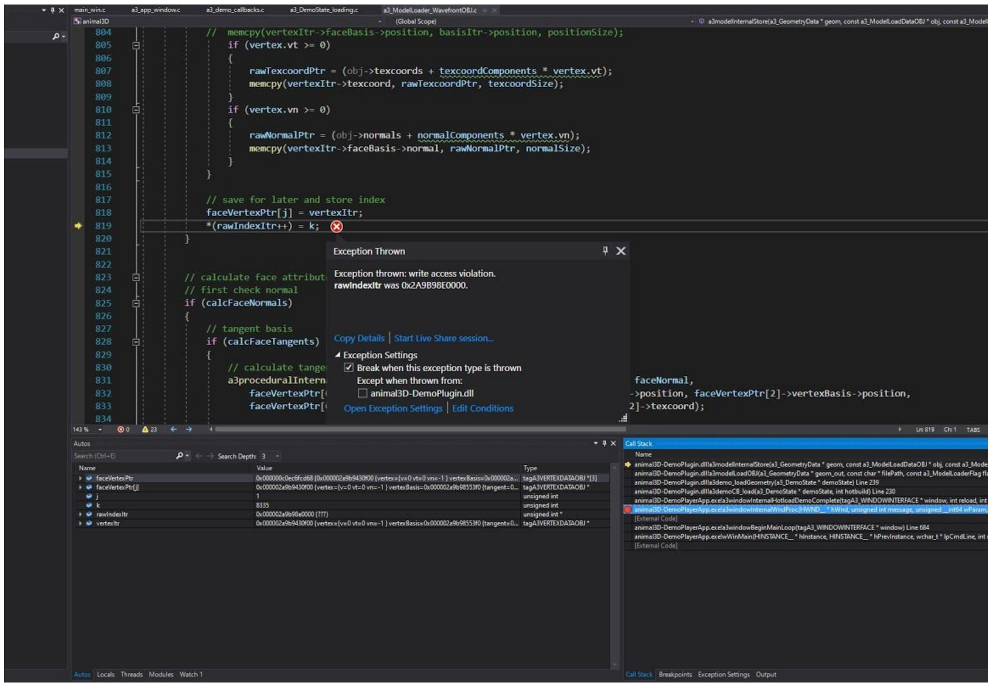
Bug 1: Vertex Perplexion

Description of Issue/Scenario

The core issue with the first bug is related to an exception throw during the loading process of a specific mesh object when the **demo state tries to load vertices for a mesh file object.**

Details of Issue/Scenario

The issue pertains to the loading of one of the mesh objects for the scene during the function a3modelInternalStore in a3_ModelLoader_WavefrontOBJ.c. When the demo state is loaded without any code changes, the program hangs for about 2 seconds and then immediately throws an exception at line 819 of a3_ModelLoader_WavefrontOBJ.c after running through a series of iterations for storing vertexes for mesh faces.


main_win.c
a3_app_window.c
a3_demo_callbacks.c
a3_DemoState_loading.c
a3_ModelLoader_WavefrontOBJ.c
animal3D
(Global Scope)
a3modelInternalStore(a3_GeometryData * geom, const a3_ModelLoadDataOBJ * obj, const a3_Model

```
804  //  memcpy(vertexItr->faceBasis->position, basisItr->position, positionSize);
805      if (vertex.vt >= 0)
806      {
807          rawTexcoordPtr = (obj->texcoords + texcoordComponents * vertex.vt);
808          memcpy(vertexItr->texcoord, rawTexcoordPtr, texcoordSize);
809      }
810      if (vertex.vn >= 0)
811      {
812          rawNormalPtr = (obj->normals + normalComponents * vertex.vn);
813          memcpy(vertexItr->faceBasis->normal, rawNormalPtr, normalSize);
814      }
815  }
816
817  // save for later and store index
818  faceVertexPtr[j] = vertexItr;
819  *(rawIndexItr++) = k;
820  }
821
822
823  // calculate face attribut
824  // first check normal
825  if (calcFaceNormals)
826  {
827      // tangent basis
828      if (calcFaceTangents)
829      {
830          // calculate tange
831          a3proceduralIntern                                     faceNormal,
832              faceVertexPtr[          ->position, faceVertexPtr[2]->vertexBasis->position,
833              faceVertexPtr[          2]->texcoord);
834
```

Exception Thrown

Exception thrown: write access violation.
rawIndexItr was 0x2A9B98E0000.

☑ Break when this exception type is thrown
Except when thrown from:
☐ animal3D-DemoPlugin.dll

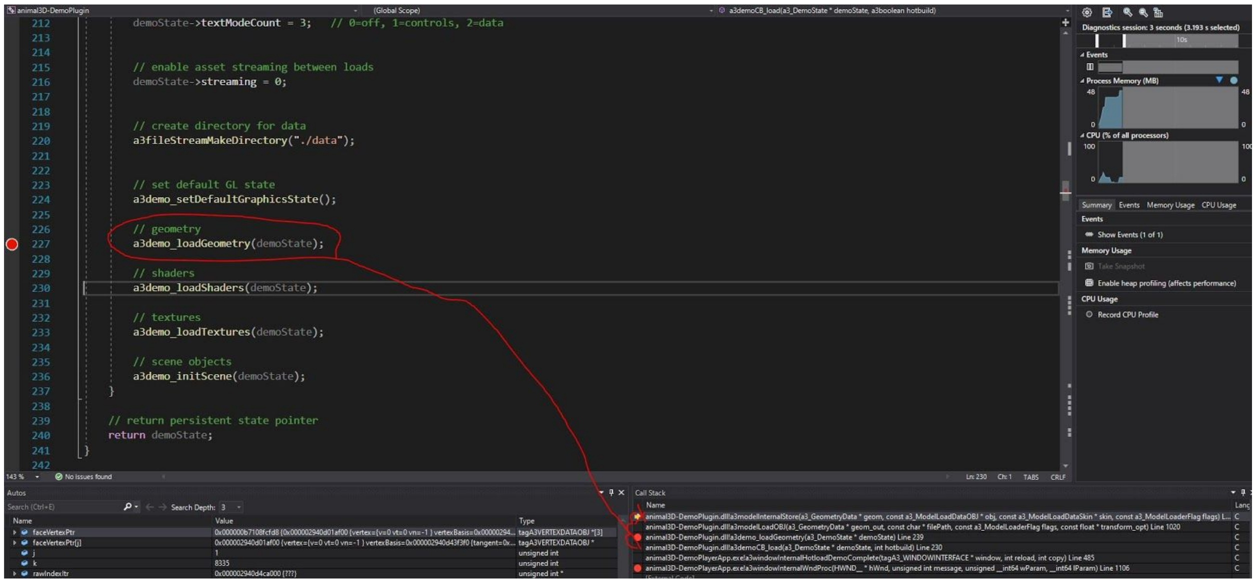
Name	Value	Type
faceVertexPtr	0x000000c0ec6fcd68 {0x000002a9b9430f00 {vertex={v=0 vt=0 vn=-1 } vertexBasis=0x000002a...	tagA3VERTEXDATAOBJ *[3]
faceVertexPtr[j]	0x000002a9b9430f00 {vertex={v=0 vt=0 vn=-1 } vertexBasis=0x000002a9b98553f0 {tangent=0...	tagA3VERTEXDATAOBJ *
j	1	unsigned int
k	8335	unsigned int
rawIndexItr	0x000002a9b98e0000 {???}	unsigned int *
vertexItr	0x000002a9b9430f00 {vertex={v=0 vt=0 vn=-1 } vertexBasis=0x000002a9b98553f0 {tangent=0...	tagA3VERTEXDATAOBJ *

Call Stack

Name
animal3D-DemoPlugin.dll!a3modelInternalStore(a3_GeometryData * geom, const a3_ModelLoadDataOBJ * obj, const a3_Mode
animal3D-DemoPlugin.dll!a3modelLoadOBJ(a3_GeometryData * geom_out, const char * filePath, const a3_ModelLoaderFlag fl
animal3D-DemoPlugin.dll!a3demo_loadGeometry(a3_DemoState * demoState) Line 239
animal3D-DemoPlugin.dll!a3demoCB_load(a3_DemoState * demoState, int hotbuild) Line 230
animal3D-DemoPlayerApp.exe!a3windowInternalHotloadDemoComplete(tagA3_WINDOWINTERFACE * window, int reload, int
animal3D-DemoPlayerApp.exe!a3windowInternalWndProc(HWND__ * hWnd, unsigned int message, unsigned __int64 wParam,
[External Code]
animal3D-DemoPlayerApp.exe!a3windowBeginMainLoop(tagA3_WINDOWINTERFACE * window) Line 684
animal3D-DemoPlayerApp.exe!wWinMain(HINSTANCE__ * hInstance, HINSTANCE__ * hPrevInstance, wchar_t * lpCmdLine, int
[External Code]

Autos Locals Threads Modules Watch 1
Call Stack Breakpoints Exception Settings Output

Looking at the call stack, this function is called out of the a3modelLoadOBJ function in the same file. If we examine the call stack in the below photo, we can trace the entire execution leading up to where the breakpoint triggers in order to narrow down what parts of the process are and aren't working:



```
		demoState->textModeCount = 3;	// 0=off, 1=controls, 2=data


		// enable asset streaming between loads
		demoState->streaming = 0;


		// create directory for data
		a3fileStreamMakeDirectory("./data");


		// set default GL state
		a3demo_setDefaultGraphicsState();

		// geometry
		a3demo_loadGeometry(demoState);

		// shaders
		a3demo_loadShaders(demoState);

		// textures
		a3demo_loadTextures(demoState);

		// scene objects
		a3demo_initScene(demoState);
	}

	// return persistent state pointer
	return demoState;
}
```

```
Autos
Name              Value                                                                                               Type
faceVertexPtr     0x000000b7108fcfd8 {0x000002940d01af00 {vertex={v=0 vt=0 vn=-1 } vertexBasis=0x00000294...       tagA3VERTEXDATAOBJ *[3]
faceVertexPtr[j]  0x000002940d01af00 {vertex={v=0 vt=0 vn=-1 } vertexBasis=0x000002940d43f3f0 {tangent=0x...      tagA3VERTEXDATAOBJ *
j                 1                                                                                                   unsigned int
k                 8335                                                                                                unsigned int
rawIndexItr       0x000002940d4ca000 {???}                                                                            unsigned int *
```

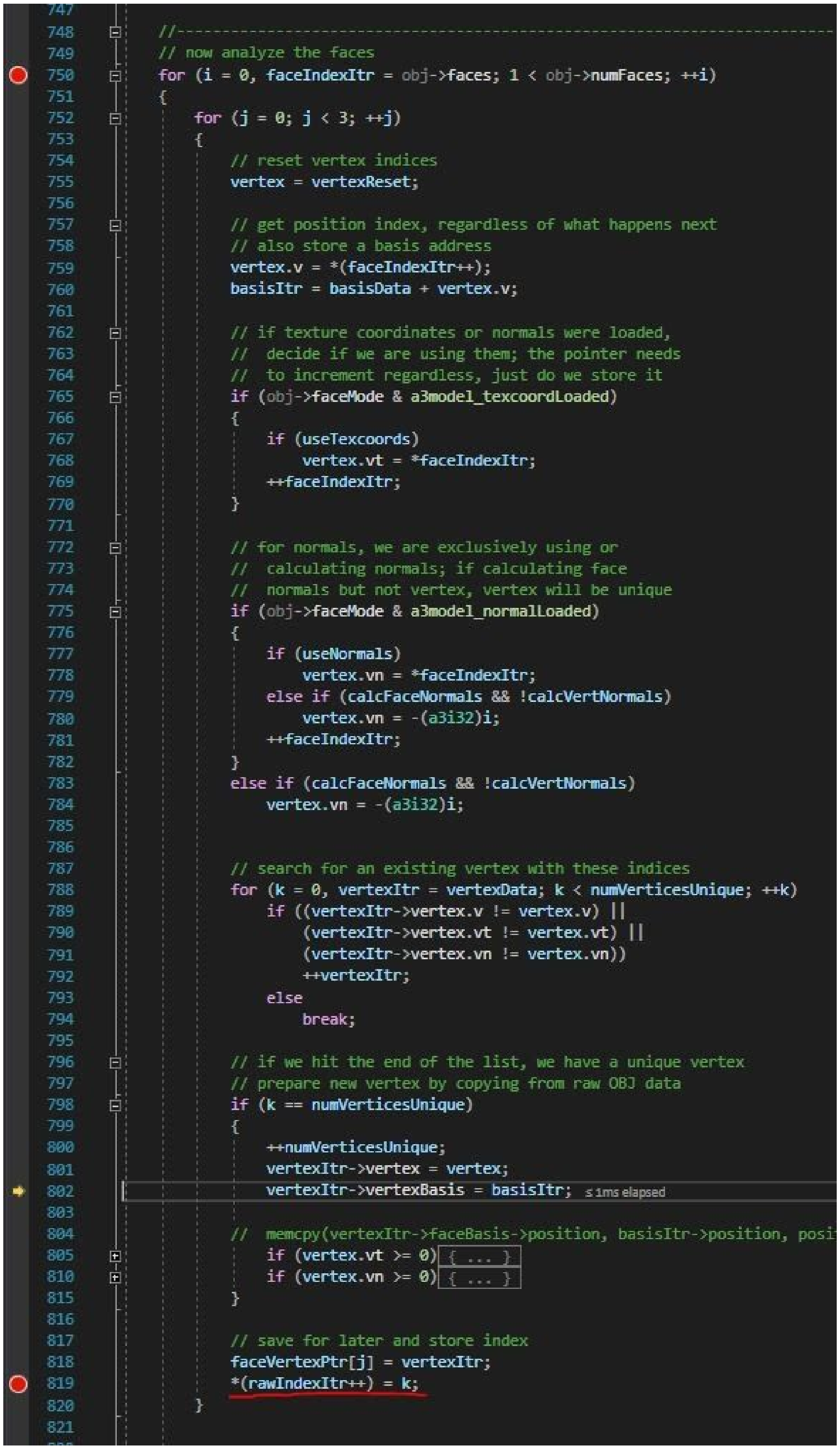
```
Call Stack
Name                                                                                                                                                                  Lang
animal3D-DemoPlugin.dll!a3modelInternalStore(a3_GeometryData * geom, const a3_ModelLoadDataOBJ * obj, const a3_ModelLoadDataSkin * skin, const a3_ModelLoaderFlag flags) L...   C
animal3D-DemoPlugin.dll!a3modelLoadOBJ(a3_GeometryData * geom_out, const char * filePath, const a3_ModelLoaderFlag flags, const float * transform_opt) Line 1020          C
animal3D-DemoPlugin.dll!a3demo_loadGeometry(a3_DemoState * demoState) Line 239                                                                                         C
animal3D-DemoPlugin.dll!a3demoCB_load(a3_DemoState * demoState, int hotbuild) Line 230                                                                                 C
animal3D-DemoPlayerApp.exe!a3windowInternalHotloadDemoComplete(tagA3_WINDOWINTERFACE * window, int reload, int copy) Line 485                                          C
animal3D-DemoPlayerApp.exe!a3windowInternalWndProc(HWND__ * hWnd, unsigned int message, unsigned __int64 wParam, __int64 lParam) Line 1106                             C
```

1. First the program goes through the main window initialization loop, which reports no issues and works properly
2. Once the demo is loaded by the user, the program calls the a3demoCB_load function in a3_demo_callbacks.
3. Inside of this function is the first call in the trace through this issue; a3demo_loadGeometry. This keys in that the issue is happening during the geometry loading phase.
4. That function eventually calls into the backend loading for geometry; a3modelLoadOBJ in a3_ModelLoader_WavefrontOBJ.c.
5. Finally, this function calls a3modelInternalStore in the same file, which is where the issue eventually happens.

What should be happening here is that the trig table and timer are successfully initialized to allow text to be initialized as well, but something is going wrong before the text is initialized that's preventing it from returning successfully. Moving the call to a3demo_initializeText above the a3trigInit call prevents the exception from happening, meaning the problem is with **one of the three lines above the actual call to the text drawing.**

What should be happening here is that the face vertex array should successfully initialize to allow the model to be drawn in the scene, but something is going wrong during the internal storage conversion to use a common data format during the face analysis process. Moving the code around and commenting out certain lines didn't temporarily stop the issue from happening, meaning the problem is more than likely related to an **overflow of the face vertex array** which is triggering the exception.

The issue is not related specifically to the model, its file path, or loading the mesh successfully but rather the fact that it cannot be stored properly as internal storage. After narrowing down every function call with agonizing detail, step ins, and breakpoints, I tracked the root of the issue to somewhere in the face analysis section of the a3modelInternalStore function; the image below shows the problem area where any one of the lines could be the source of the issue:



```
//---------------------------------------------------------------------------
// now analyze the faces
for (i = 0, faceIndexItr = obj->faces; 1 < obj->numFaces; ++i)
{
    for (j = 0; j < 3; ++j)
    {
        // reset vertex indices
        vertex = vertexReset;

        // get position index, regardless of what happens next
        // also store a basis address
        vertex.v = *(faceIndexItr++);
        basisItr = basisData + vertex.v;

        // if texture coordinates or normals were loaded,
        //  decide if we are using them; the pointer needs
        //  to increment regardless, just do we store it
        if (obj->faceMode & a3model_texcoordLoaded)
        {
            if (useTexcoords)
                vertex.vt = *faceIndexItr;
            ++faceIndexItr;
        }

        // for normals, we are exclusively using or
        //  calculating normals; if calculating face
        //  normals but not vertex, vertex will be unique
        if (obj->faceMode & a3model_normalLoaded)
        {
            if (useNormals)
                vertex.vn = *faceIndexItr;
            else if (calcFaceNormals && !calcVertNormals)
                vertex.vn = -(a3i32)i;
            ++faceIndexItr;
        }
        else if (calcFaceNormals && !calcVertNormals)
            vertex.vn = -(a3i32)i;


        // search for an existing vertex with these indices
        for (k = 0, vertexItr = vertexData; k < numVerticesUnique; ++k)
            if ((vertexItr->vertex.v != vertex.v) ||
                (vertexItr->vertex.vt != vertex.vt) ||
                (vertexItr->vertex.vn != vertex.vn))
                ++vertexItr;
            else
                break;

        // if we hit the end of the list, we have a unique vertex
        // prepare new vertex by copying from raw OBJ data
        if (k == numVerticesUnique)
        {
            ++numVerticesUnique;
            vertexItr->vertex = vertex;
            vertexItr->vertexBasis = basisItr;  ≤ 1ms elapsed

        //  memcpy(vertexItr->faceBasis->position, basisItr->position, posi
            if (vertex.vt >= 0) { ... }
            if (vertex.vn >= 0) { ... }
        }

        // save for later and store index
        faceVertexPtr[j] = vertexItr;
        *(rawIndexItr++) = k;
    }

```

Discovery/Replication of Issue

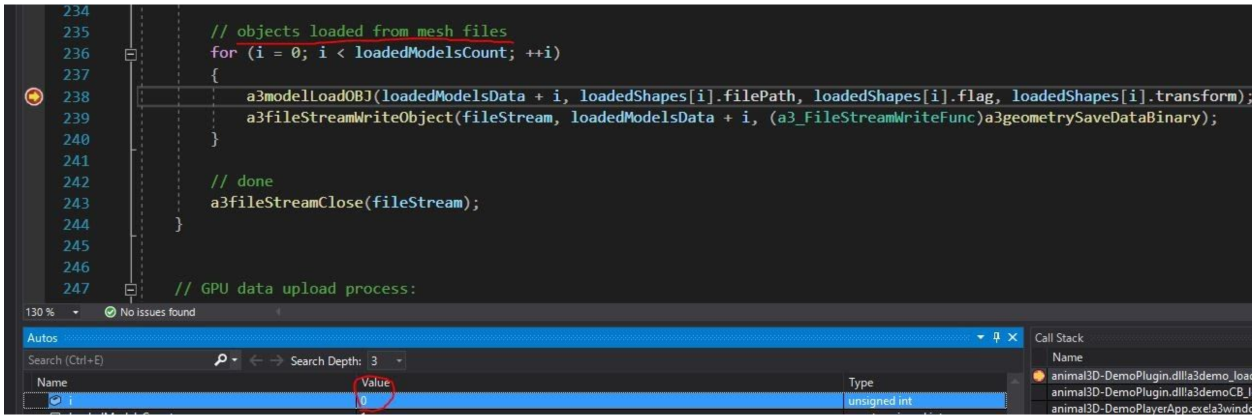
Since this was the first (and sadly only :() issue found with the program, its discovery process was extremely simple. Running the program in debug mode and selecting “DEBUG: Demo project hot build and load -> Load without building” without any alterations to the existing code causes the exception to throw after about a 2 second hanging time where the program is unresponsive. This same process can be done to replicate the bug: do not change any code and leave it as supplied, then run it to see the exception throw.

Discovering that the issue was related to the loading of an external mesh file was achieved by placing a breakpoint at the exact point where the exception is thrown and tracing the call stack backwards until the callbacks file was reached. Then, a breakpoint was placed inside the a3demoCB_load function in a3_demo_callbacks to see which of the loading functions was the start point of the issue. To replicate, place a breakpoint on the call to a3demo_loadGeometry in a3demoCB_load function in a3_demo_callbacks, then run the program, then select “DEBUG: Demo project hot build and load -> Load without building” and follow the program through until the exception is thrown.

Debugging Process

The following process was used to try and discover why this issue was happening. The process involves using a mixture of breakpoints, watches on the mesh data variables, checking mesh file data, checking iterator values, and reading through the function comments to try and decipher what they’re supposed to be doing.

First, I decided to check right inside of a3demo_loadGeometry since thats the first actual function that handles the issue with geometry. I inspected the “objects loaded from mesh files” area of this function since it is where the the call to the backend is taking place. Here I examined the iterator to see which mesh file was causing the issue; the issue always triggered when i was equal to zero, meaning it was the first mesh being loaded.

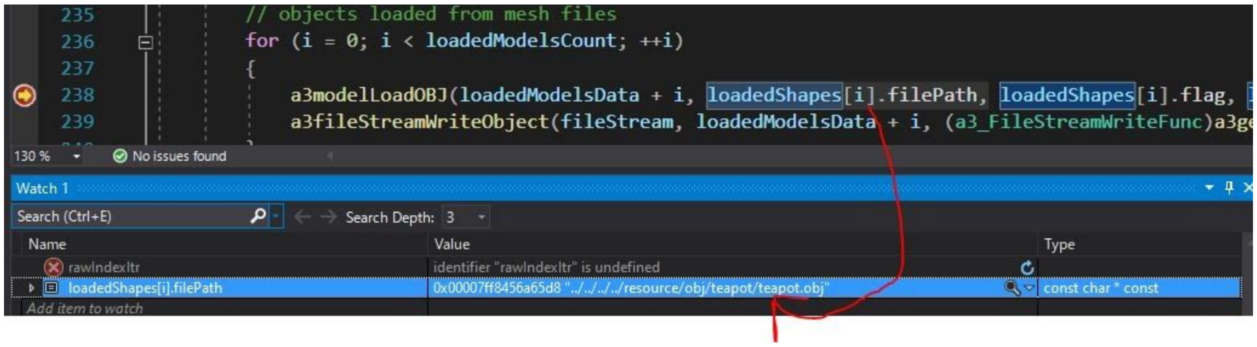


```
		// objects loaded from mesh files
		for (i = 0; i < loadedModelsCount; ++i)
		{
			a3modelLoadOBJ(loadedModelsData + i, loadedShapes[i].filePath, loadedShapes[i].flag, loadedShapes[i].transform);
			a3fileStreamWriteObject(fileStream, loadedModelsData + i, (a3_FileStreamWriteFunc)a3geometrySaveDataBinary);
		}

		// done
		a3fileStreamClose(fileStream);
	}


	// GPU data upload process:
```

```
Autos
Name        Value       Type
i           0           unsigned int
```

```
Call Stack
Name
animal3D-DemoPlugin.dll!a3demo_loa
animal3D-DemoPlugin.dll!a3demoCB_l
animal3D-DemoPlayerApp.exe!a3windo
```

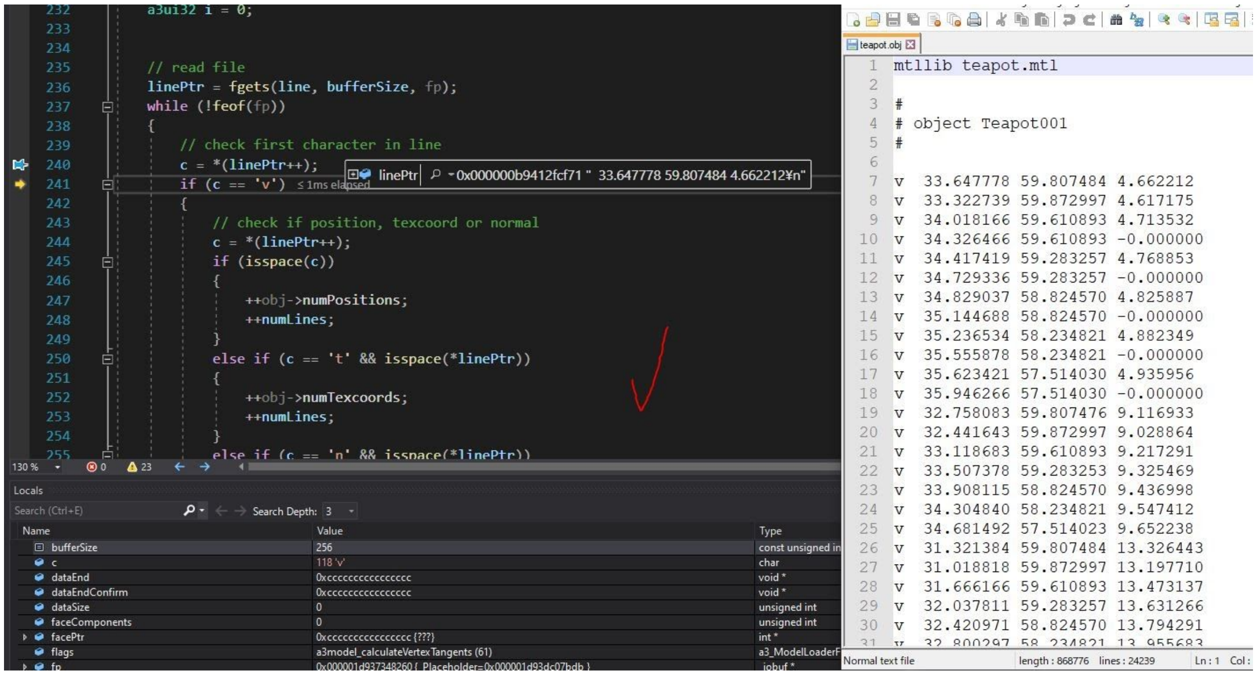
After discovering that the first mesh is the one causing the issue, I had a thought that perhaps the file name was not being set correctly so I decided to inspect that with a watch; turns out the file path was indeed correct and the mesh being loaded was the teapot! This, unfortunately, was not the cause of the issue, however, as the exception is still triggered.



```
		// objects loaded from mesh files
		for (i = 0; i < loadedModelsCount; ++i)
		{
			a3modelLoadOBJ(loadedModelsData + i, loadedShapes[i].filePath, loadedShapes[i].flag, 
			a3fileStreamWriteObject(fileStream, loadedModelsData + i, (a3_FileStreamWriteFunc)a3ge
```

```
130 %    No issues found

Watch 1
Search (Ctrl+E)    Search Depth: 3
Name                           Value                                                              Type
rawIndexItr                    identifier "rawIndexItr" is undefined
loadedShapes[i].filePath       0x00007ff8456a65d8 "../../../../resource/obj/teapot/teapot.obj"    const char * const
Add item to watch
```

Around this point I started to become baffled as to what was actually causing this issue. I had looked through the function documentation and traced through every step of the mesh loading process and everything was returning correctly. I had another thought; what if the file wasn't being read correctly? I decided to cross-reference the contents of the teapot mesh file with what was being read into the file stream in a3modelLoadOBJ. This was another dead end, as the contents of the file were being read correctly...



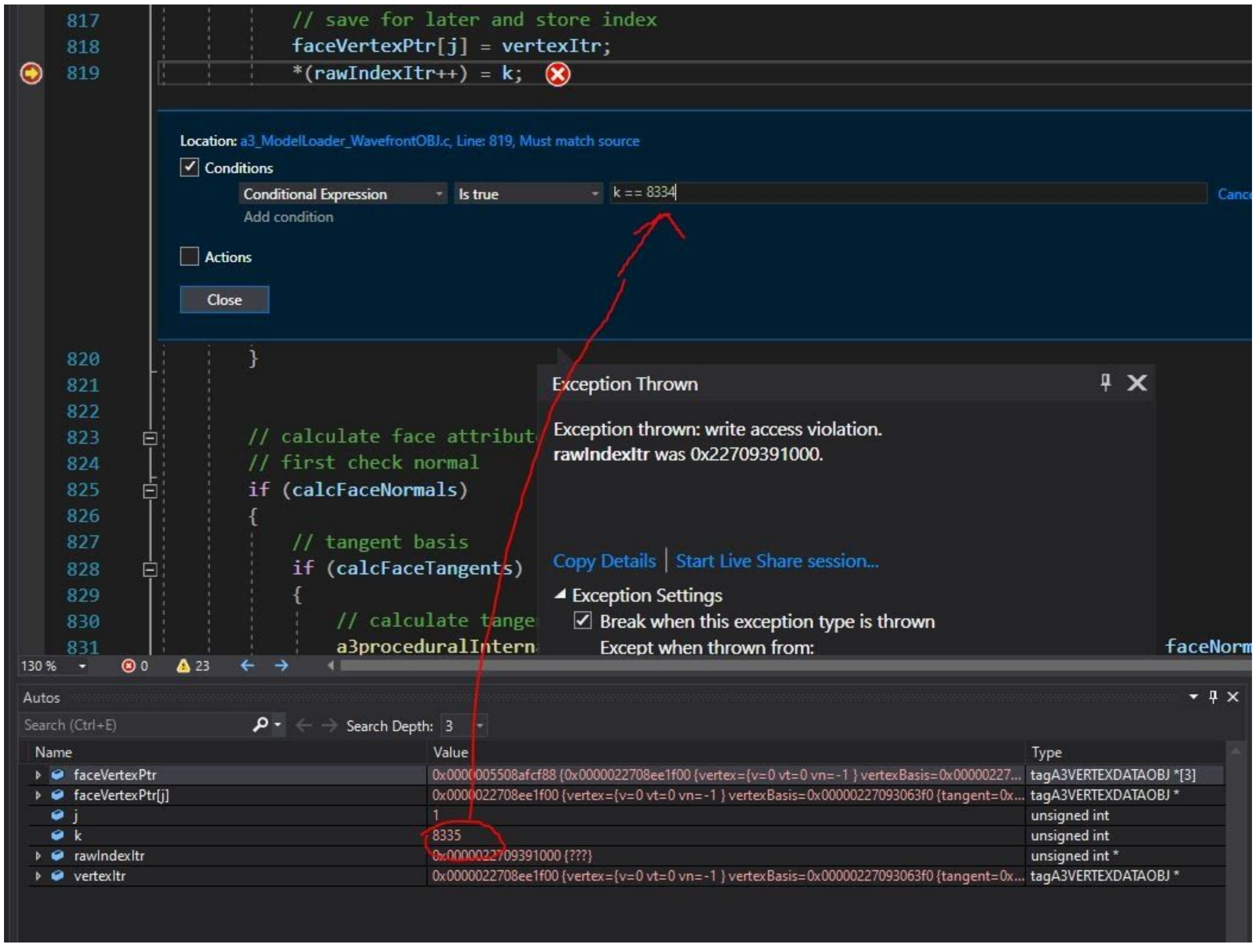
```
	a3ui32 i = 0;


	// read file
	linePtr = fgets(line, bufferSize, fp);
	while (!feof(fp))
	{
		// check first character in line
		c = *(linePtr++);
		if (c == 'v')  ≤ 1ms elapsed
		{
			// check if position, texcoord or normal
			c = *(linePtr++);
			if (isspace(c))
			{
				++obj->numPositions;
				++numLines;
			}
			else if (c == 't' && isspace(*linePtr))
			{
				++obj->numTexcoords;
				++numLines;
			}
			else if (c == 'n' && isspace(*linePtr))
```

```
linePtr | 0x000000b9412fcf71 " 33.647778 59.807484 4.662212¥n"
```

```
130 %    0    23

Locals
Search (Ctrl+E)    Search Depth: 3
Name               Value                                                     Type
bufferSize         256                                                       const unsigned in
c                  118 'v'                                                   char
dataEnd            0xcccccccccccccccc                                        void *
dataEndConfirm     0xcccccccccccccccc                                        void *
dataSize           0                                                         unsigned int
faceComponents     0                                                         unsigned int
facePtr            0xcccccccccccccccc {???}                                  int *
flags              a3model_calculateVertexTangents (61)                      a3_ModelLoaderF
fp                 0x000001d937348260 { Placeholder=0x000001d93dc07bdb }     iobuf *
```

```
teapot.obj
mtllib teapot.mtl

#
# object Teapot001
#

v  33.647778 59.807484 4.662212
v  33.322739 59.872997 4.617175
v  34.018166 59.610893 4.713532
v  34.326466 59.610893 -0.000000
v  34.417419 59.283257 4.768853
v  34.729336 59.283257 -0.000000
v  34.829037 58.824570 4.825887
v  35.144688 58.824570 -0.000000
v  35.236534 58.234821 4.882349
v  35.555878 58.234821 -0.000000
v  35.623421 57.514030 4.935956
v  35.946266 57.514030 -0.000000
v  32.758083 59.807476 9.116933
v  32.441643 59.872997 9.028864
v  33.118683 59.610893 9.217291
v  33.507378 59.283253 9.325469
v  33.908115 58.824570 9.436998
v  34.304840 58.234821 9.547412
v  34.681492 57.514023 9.652238
v  31.321384 59.807484 13.326443
v  31.018818 59.872997 13.197710
v  31.666166 59.610893 13.473137
v  32.037811 59.283257 13.631266
v  32.420971 58.824570 13.794291
v  32.800297 58.234821 13.955683
Normal text file    length : 868776   lines : 24239    Ln : 1   Col :
```

Perplexed and confused, I decided to attempt running through the iterations in the face analyzation section of a3modelInternalStore. This was a *horrible* idea, as I soon realized the loop runs many, many times...8335 times to be exact. Once the iteration hits this number, the exception throws, so I decided to try adding a conditional breakpoint to the exception line to break when the iteration is one tick before the break of 8335.



```
            // save for later and store index
            faceVertexPtr[j] = vertexItr;
            *(rawIndexItr++) = k;
```

Location: a3_ModelLoader_WavefrontOBJ.c, Line: 819, Must match source

☑ Conditions

Conditional Expression | Is true | k == 8334

Add condition

☐ Actions

Close

```
        }


        // calculate face attribut
        // first check normal
        if (calcFaceNormals)
        {
            // tangent basis
            if (calcFaceTangents)
            {
                // calculate tange
                a3proceduralIntern
```

Exception Thrown

Exception thrown: write access violation.
rawIndexItr was 0x22709391000.

Copy Details | Start Live Share session...

Exception Settings

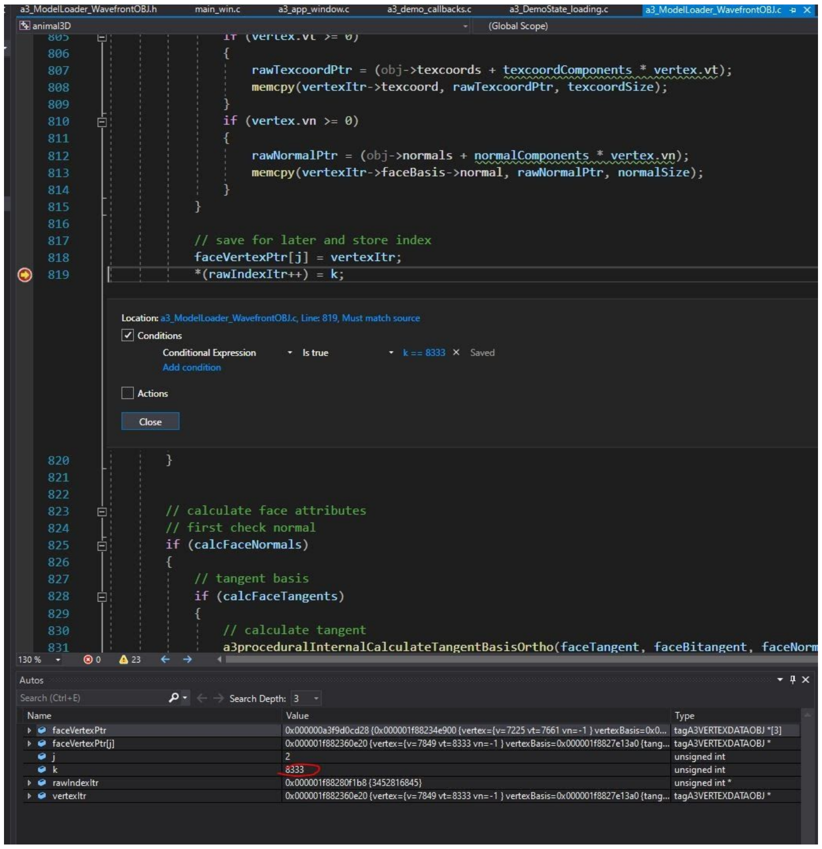
☑ Break when this exception type is thrown

Except when thrown from:

Autos

Name	Value	Type
faceVertexPtr	0x0000005508afcf88 {0x0000022708ee1f00 {vertex={v=0 vt=0 vn=-1 } vertexBasis=0x00000227...	tagA3VERTEXDATAOBJ *[3]
faceVertexPtr[j]	0x0000022708ee1f00 {vertex={v=0 vt=0 vn=-1 } vertexBasis=0x00000227093063f0 {tangent=0x...	tagA3VERTEXDATAOBJ *
j	1	unsigned int
k	8335	unsigned int
rawIndexItr	0x0000022709391000 {???}	unsigned int *
vertexItr	0x0000022708ee1f00 {vertex={v=0 vt=0 vn=-1 } vertexBasis=0x00000227093063f0 {tangent=0x...	tagA3VERTEXDATAOBJ *

I executed the program with this new breakpoint and loaded the demo state...and it just sat there. For about 30 seconds, the program window was completely hung and frozen. I couldn't move the window around or do anything to it. Then finally the breakpoint hit and let me inspect what was happening with the faceVertexPtr.



```
                if (vertex.vt >= 0)
                {
                    rawTexcoordPtr = (obj->texcoords + texcoordComponents * vertex.vt);
                    memcpy(vertexItr->texcoord, rawTexcoordPtr, texcoordSize);
                }
                if (vertex.vn >= 0)
                {
                    rawNormalPtr = (obj->normals + normalComponents * vertex.vn);
                    memcpy(vertexItr->faceBasis->normal, rawNormalPtr, normalSize);
                }
            }

            // save for later and store index
            faceVertexPtr[j] = vertexItr;
            *(rawIndexItr++) = k;
```

Location: a3_ModelLoader_WavefrontOBJ.c, Line: 819, Must match source

☑ Conditions

Conditional Expression | Is true | k == 8333 Saved

Add condition

☐ Actions

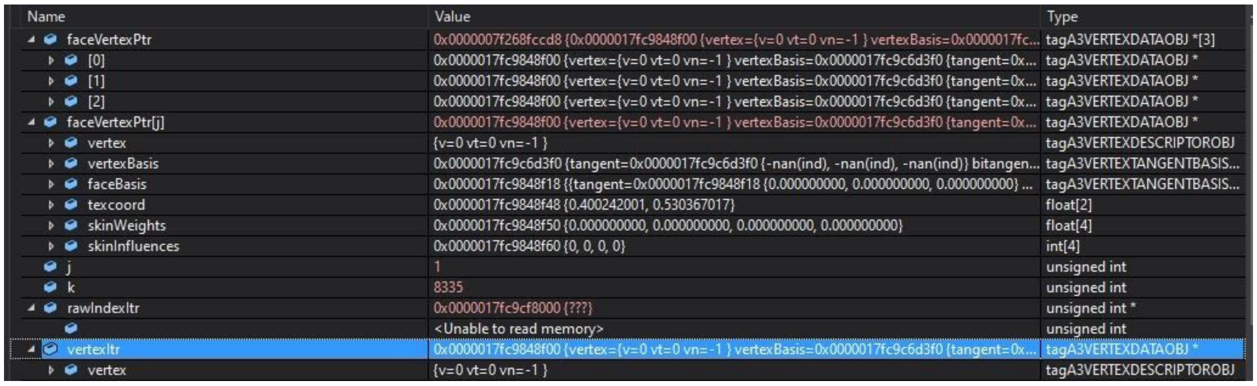
Close

```
        }


        // calculate face attributes
        // first check normal
        if (calcFaceNormals)
        {
            // tangent basis
            if (calcFaceTangents)
            {
                // calculate tangent
                a3proceduralInternalCalculateTangentBasisOrtho(faceTangent, faceBitangent, faceNorm
```

Autos

Name	Value	Type
faceVertexPtr	0x000000a3f9d0cd28 {0x000001f88234e900 {vertex={v=7225 vt=7661 vn=-1 } vertexBasis=0x0...	tagA3VERTEXDATAOBJ *[3]
faceVertexPtr[j]	0x000001f882360e20 {vertex={v=7849 vt=8333 vn=-1 } vertexBasis=0x000001f8827e13a0 {tang...	tagA3VERTEXDATAOBJ *
j	2	unsigned int
k	8333	unsigned int
rawIndexItr	0x000001f88280f1b8 {3452816845}	unsigned int *
vertexItr	0x000001f882360e20 {vertex={v=7849 vt=8333 vn=-1 } vertexBasis=0x000001f8827e13a0 {tang...	tagA3VERTEXDATAOBJ *

Inspecting the faceVertexPtr reveals what I've concluded is the source of the issue but am unsure of the root cause; the vertices inside the array are set to 0, 0, and -1 respectively when the inspection throws and the rawIndex ptr becomes null and unreadable. I think that the vertex are being overloaded somewhere and causing the program to hang for so long that it eventually fails and throws an exception.



Name	Value	Type
faceVertexPtr	0x0000007f268fccd8 {0x0000017fc9848f00 {vertex={v=0 vt=0 vn=-1 } vertexBasis=0x0000017fc...	tagA3VERTEXDATAOBJ *[3]
[0]	0x0000017fc9848f00 {vertex={v=0 vt=0 vn=-1 } vertexBasis=0x0000017fc9c6d3f0 {tangent=0x...	tagA3VERTEXDATAOBJ *
[1]	0x0000017fc9848f00 {vertex={v=0 vt=0 vn=-1 } vertexBasis=0x0000017fc9c6d3f0 {tangent=0x...	tagA3VERTEXDATAOBJ *
[2]	0x0000017fc9848f00 {vertex={v=0 vt=0 vn=-1 } vertexBasis=0x0000017fc9c6d3f0 {tangent=0x...	tagA3VERTEXDATAOBJ *
faceVertexPtr[j]	0x0000017fc9848f00 {vertex={v=0 vt=0 vn=-1 } vertexBasis=0x0000017fc9c6d3f0 {tangent=0x...	tagA3VERTEXDATAOBJ *
vertex	{v=0 vt=0 vn=-1 }	tagA3VERTEXDESCRIPTOROBJ
vertexBasis	0x0000017fc9c6d3f0 {tangent=0x0000017fc9c6d3f0 {-nan(ind), -nan(ind), -nan(ind)} bitangen...	tagA3VERTEXTANGENTBASIS...
faceBasis	0x0000017fc9848f18 {{tangent=0x0000017fc9848f18 {0.000000000, 0.000000000, 0.000000000} ...	tagA3VERTEXTANGENTBASIS...
texcoord	0x0000017fc9848f48 {0.400242001, 0.530367017}	float[2]
skinWeights	0x0000017fc9848f50 {0.000000000, 0.000000000, 0.000000000, 0.000000000}	float[4]
skinInfluences	0x0000017fc9848f60 {0, 0, 0, 0}	int[4]
j	1	unsigned int
k	8335	unsigned int
rawIndexItr	0x0000017fc9cf8000 {???}	unsigned int *
	<Unable to read memory>	unsigned int
vertexItr	0x0000017fc9848f00 {vertex={v=0 vt=0 vn=-1 } vertexBasis=0x0000017fc9c6d3f0 {tangent=0x...	tagA3VERTEXDATAOBJ *
vertex	{v=0 vt=0 vn=-1 }	tagA3VERTEXDESCRIPTOROBJ

Hypothetical Solution

Based on the findings from the discovery and debugging process, I propose the following hypothetical solution.

Since the faceVertexPtr values are being reset to null values after 8335 iterations of the loop, I would continue running through the program with breakpoints to find the value controlling how many vertices are allowed to be stored inside of the vertex ptr array. What I think is happening is that the **program is allocating too much space for this mesh** which is causing it to hang and crash. I think looking for the size controller of the mesh pointer would be useful because it would allow us to **modify how many vertexes can be loaded into the variable** and see if that affects the load time/exception throwing process.