

Project 2: Real Time Bug Hunting Report

Bug 1: Initializing Failure	1
Description of Issue/Scenario	1
Details of Issue/Scenario	2
Discovery/Replication of Issue	4
Debugging Process	4
Solution	6
Bug 2: Geometry/Vertex Format Mismatch	7
Description of Issue/Scenario	7
Details of Issue/Scenario	7
Discovery/Replication of Issue	9
Debugging Process	9
Solution	11
Bug 3: Refusing to Render	13
Description of Issues/Scenarios	13
Details of Issues/Scenarios	13
Discovery/Replication of Issues	13
Debugging Process	14
Solution	16
Bug 4: Rotation Frustration	17
Description of Issues/Scenarios	17
Details of Issue/Scenario	17
Discovery/Replication of Issues	18
Debugging Process	18
Solution	20

Bug 1: Initializing Failure

Description of Issue/Scenario

The core issue with the first bug is related to an exception throw during the initialization process of the demo state when the demo state tries to **initialize text drawing functionality**.

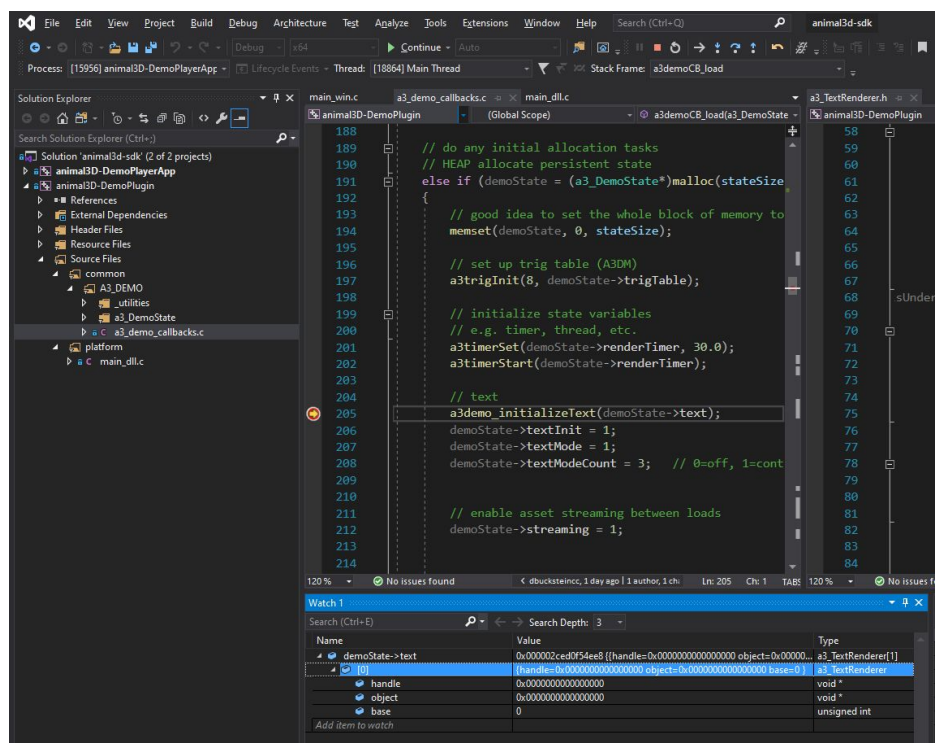
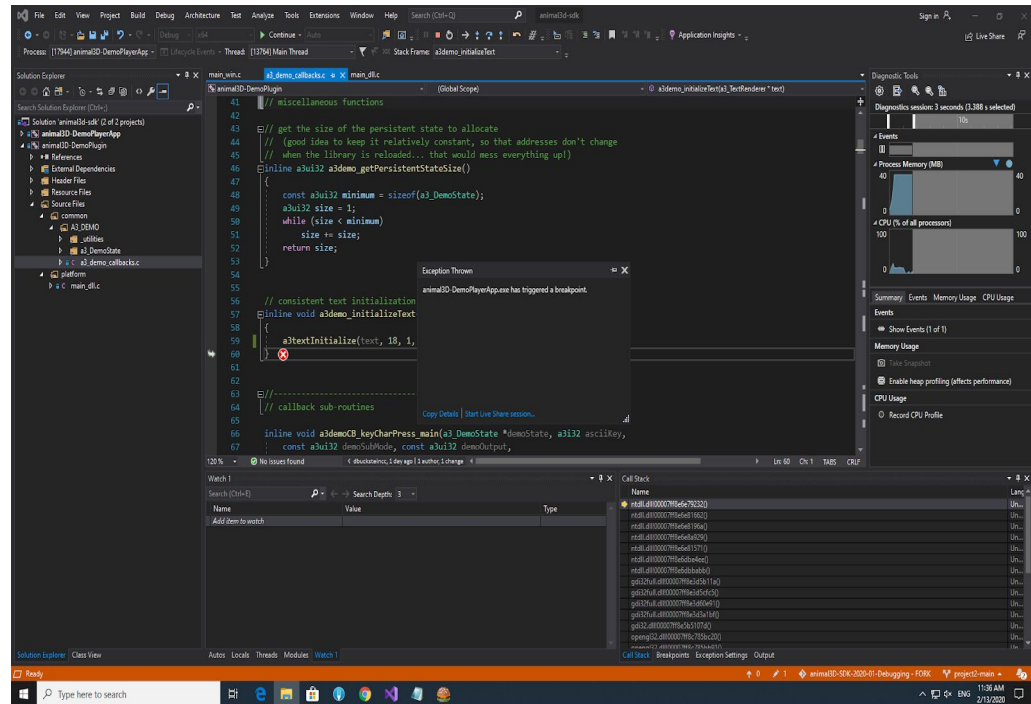
Details of Issue/Scenario

The issue pertains to the initialization of the demo state during the function `a3demoCB_Load` in `a3_demo_callbacks.c`. When the demo state is loaded without any changes to the code, the program immediately throws an exception at line 60 of `a3_demo_callbacks.c` after attempting to call `a3textInitialize`, which is visible to the right.

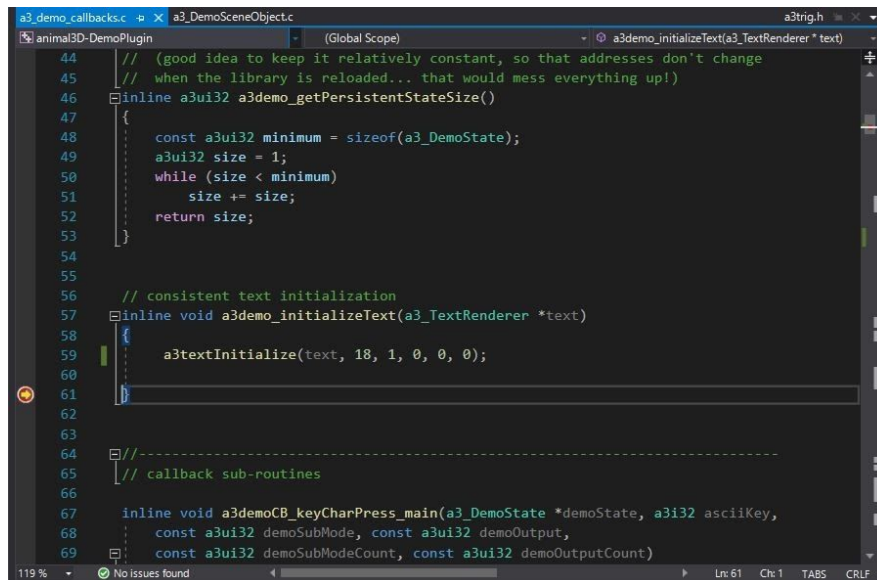
Looking at the call stack, this function is called out of the `a3demoCB_Load` function. Before `a3_demo_initializeText` is called, three other initialization to the demo state are performed:

1. `a3TrigInit` is called to initialize trigonometry data for the models
2. `a3TimerSet` is called to initialize the framerate loop timer.
3. `a3TimerStart` is called to begin the timer update

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**.

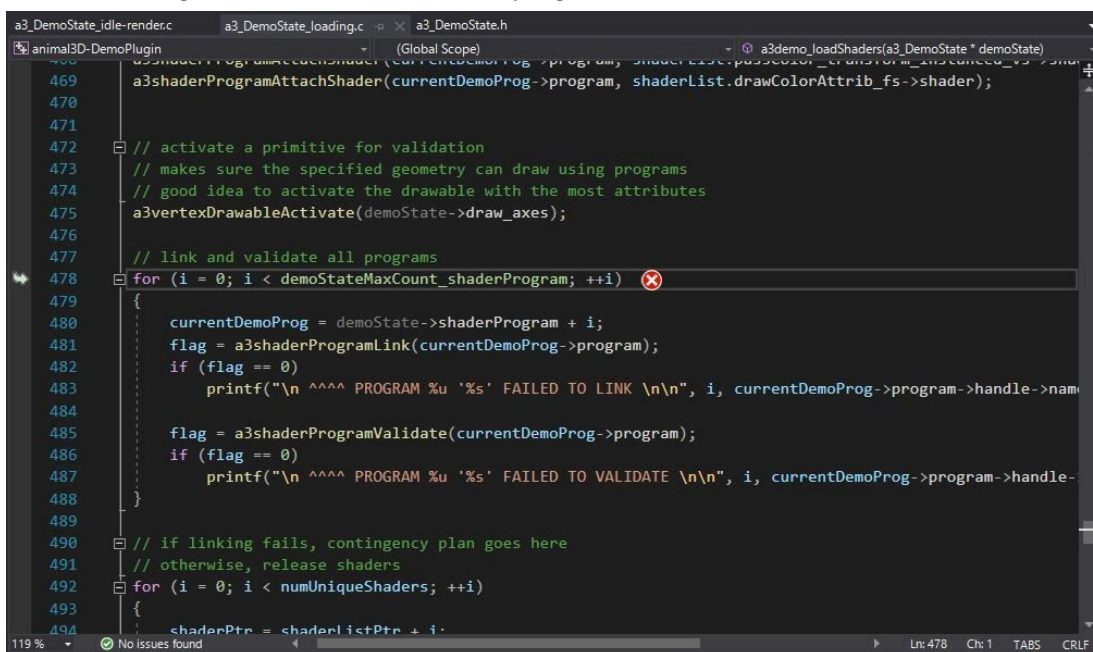


The issue is not related specifically to the text itself; in the following image, the program successfully reaches the point at the end of the function, meaning the text has not thrown the exception upon initialization.



```
a3_demo_callbacks.c | a3_DemoSceneObject.c | a3trig.h
animal3D-DemoPlugin | (Global Scope) | a3demo_initializeText(a3_TextRenderer *text)
44 // (good idea to keep it relatively constant, so that addresses don't change
45 // when the library is reloaded... that would mess everything up!)
46 inline a3ui32 a3demo_getPersistentStateSize()
47 {
48     const a3ui32 minimum = sizeof(a3_DemoState);
49     a3ui32 size = 1;
50     while (size < minimum)
51         size += size;
52     return size;
53 }
54
55
56 // consistent text initialization
57 inline void a3demo_initializeText(a3_TextRenderer *text)
58 {
59     a3textInitialize(text, 18, 1, 0, 0, 0);
60 }
61
62
63
64 //-----
65 // callback sub-routines
66
67 inline void a3demoCB_keyCharPress_main(a3_DemoState *demoState, a3i32 asciiKey,
68     const a3ui32 demoSubMode, const a3ui32 demoOutput,
69     const a3ui32 demoSubModeCount, const a3ui32 demoOutputCount)
```

However, doing this creates a new issue that points to a different area of the program; the idle render loop. A new exception occurs at a3_DemoState_idle-render.c at line 478, which is where the program tries to successfully draw the geometry when idling. This further serves to indicate that the issue isn't about the text, but rather with the **initialization of whatever is loading the geometry in the demo state**. Since the renderTimer related functions aren't related specifically to memory management, it can be assumed that the line causing this issue is related to the **a3trigInit** call, since the table is supposed to store all the trig values that the rest of the program needs.

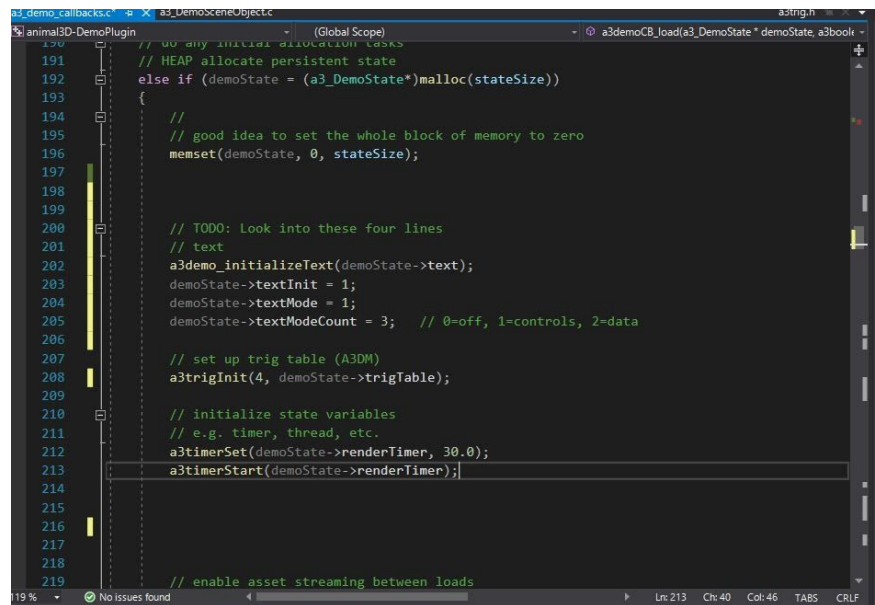


```
a3_DemoState_idle-render.c | a3_DemoState_loading.c | a3_DemoState.h
animal3D-DemoPlugin | (Global Scope) | a3demo_loadShaders(a3_DemoState *demoState)
469 a3shaderProgramAttachShader(currentDemoProg->program, shaderList.drawColorAttrib_fs->shader);
470
471
472 // activate a primitive for validation
473 // makes sure the specified geometry can draw using programs
474 // good idea to activate the drawable with the most attributes
475 a3vertexDrawableActivate(demoState->draw_axes);
476
477 // link and validate all programs
478 for (i = 0; i < demoStateMaxCount_shaderProgram; ++i)
479 {
480     currentDemoProg = demoState->shaderProgram + i;
481     flag = a3shaderProgramLink(currentDemoProg->program);
482     if (flag == 0)
483         printf("\n ^^^^ PROGRAM %u '%s' FAILED TO LINK \n\n", i, currentDemoProg->program->handle->name);
484
485     flag = a3shaderProgramValidate(currentDemoProg->program);
486     if (flag == 0)
487         printf("\n ^^^^ PROGRAM %u '%s' FAILED TO VALIDATE \n\n", i, currentDemoProg->program->handle->name);
488 }
489
490 // if linking fails, contingency plan goes here
491 // otherwise, release shaders
492 for (i = 0; i < numUniqueShaders; ++i)
493 {
494     shaderPtr = shaderListPtr + i;
```

Discovery/Replication of Issue

Since this was the first issue found with the program, its discovery process was rather simple. Running the program in debug mode and selecting “DEBUG: Demo project hot build and load -> Load without building” without changing the original code causes the exception to throw instantly. 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 events before the text initialization were causing the issue was achieved by moving all of the code used for initializing text in `a3demoCB_Load` above the call to `trigInit` as shown below. To replicate, do this and run the program, then select “DEBUG: Demo project hot build and load -> Load without building” and observe how the exception is no longer thrown at `initializeText` but rather in the idle-render function in `a3_demoState>Loading`:



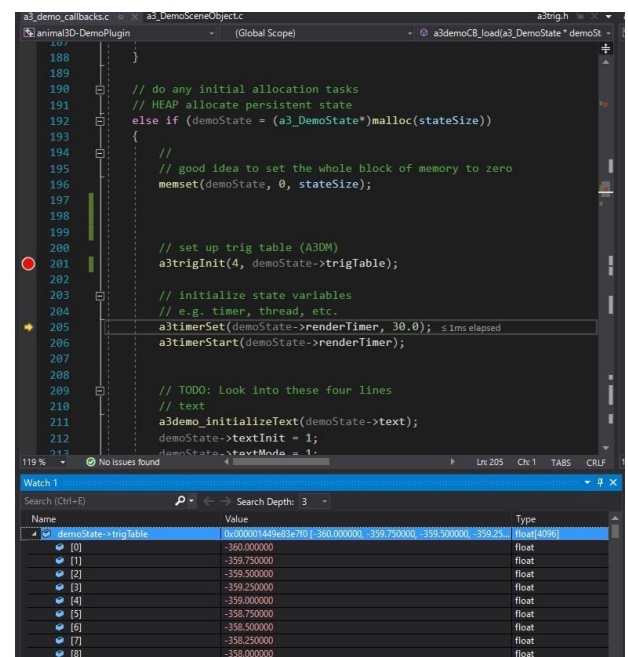
```
190 // do any initial allocation tasks
191 // HEAP allocate persistent state
192 else if (demoState = (a3_DemoState*)malloc(stateSize))
193 {
194     //
195     // good idea to set the whole block of memory to zero
196     memset(demoState, 0, stateSize);
197
198     // TODO: Look into these four lines
199     // text
200 a3demo_initializeText(demoState->text);
201 demoState->textInit = 1;
202 demoState->textMode = 1;
203 demoState->textModeCount = 3; // 0=off, 1=controls, 2=data
204
205 // set up trig table (A3DM)
206 a3trigInit(4, demoState->trigTable);
207
208 // initialize state variables
209 // e.g. timer, thread, etc.
210 a3timerSet(demoState->renderTimer, 30.0);
211 a3timerStart(demoState->renderTimer);
212
213 // enable asset streaming between loads
```

Debugging Process

The following process was used to discover why this issue was happening. The process involves using Watches on the `demoState` variables (mainly `demoState->trigTable`), setting test variables to equal the calls to see what the initialization functions are actually returning and placing Watches on them, and finally cross-referencing the results of the Watches with the documentation for the functions by looking at their definitions.

First, I returned the code to its original state after experimenting with moving around the `textInitialize` call that I described earlier in order to make sure nothing else was being modified from the original issue. I then placed a breakpoint on the call to `trigInit` and created a Watch for `demoState->trigTable` to see what was actually being stored in the table.

After seeing the results, I went to the definition for `a3trigInit` to see how it worked. The function description explains that the table is initialized as an array with that is

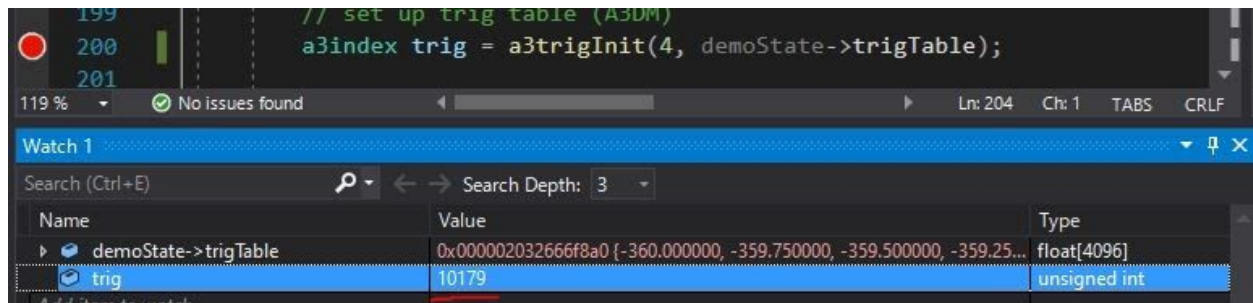


```
198 }
199
200 // do any initial allocation tasks
201 // HEAP allocate persistent state
202 else if (demoState = (a3_DemoState*)malloc(stateSize))
203 {
204     //
205     // good idea to set the whole block of memory to zero
206     memset(demoState, 0, stateSize);
207
208     // set up trig table (A3DM)
209 a3trigInit(4, demoState->trigTable);
210
211 // initialize state variables
212 // e.g. timer, thread, etc.
213 a3timerSet(demoState->renderTimer, 30.0);
214 a3timerStart(demoState->renderTimer);
215
216 // TODO: Look into these four lines
217 // text
218 a3demo_initializeText(demoState->text);
219 demoState->textInit = 1;
220 demoState->textMode = 1;
221
222 // enable asset streaming between loads
```

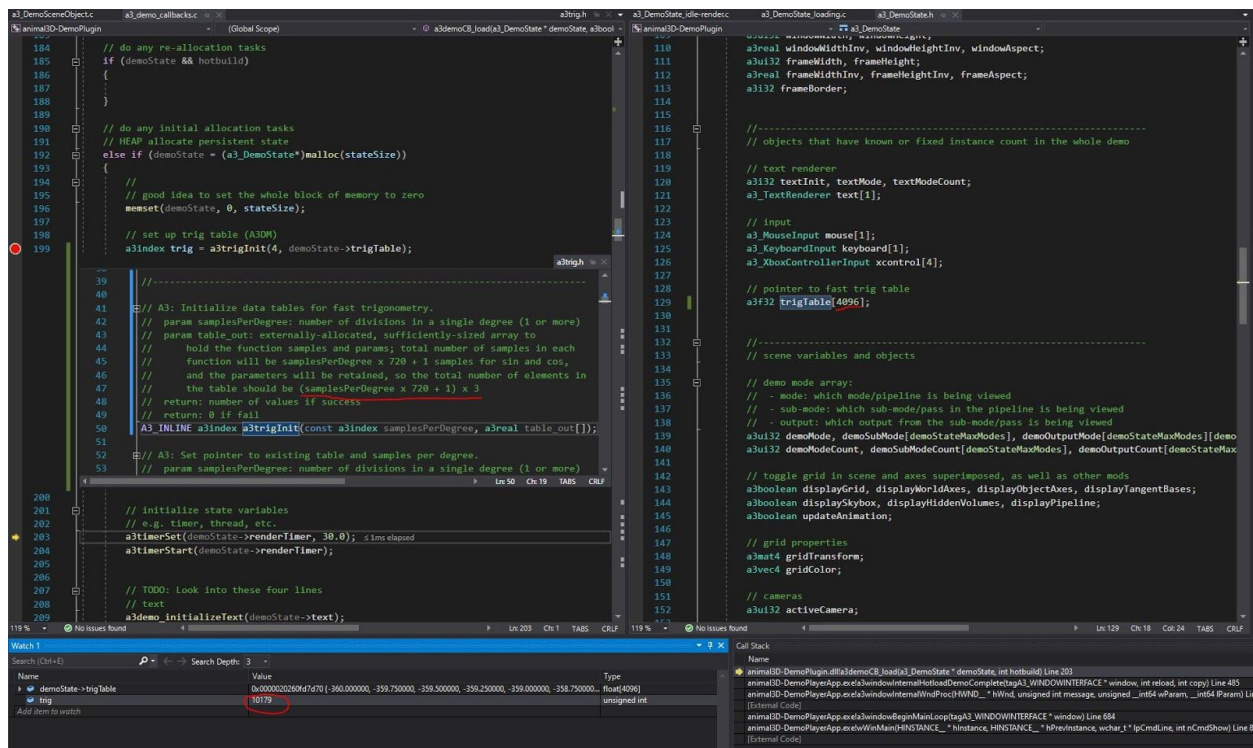
Name	Value	Type
demoState->trigTable	{0.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000}	float[9]
[0]	-359.500000	float
[1]	-359.500000	float
[2]	-359.500000	float
[3]	-359.500000	float
[4]	-359.500000	float
[5]	-359.500000	float
[6]	-359.500000	float
[7]	-359.500000	float
[8]	-359.500000	float

used to hold the samples and programs for trig functions needed by the program. This can be seen in the Watch once `a3trigInit` successfully returns; 4096 elements are initialized in `demoState->trigTable`. This size of 4096 is declared in the header for `demoState` at line 129:

When looking at the definition for the `trigInit` function, I noted that the return value will be equal to the number of trig values initialized if successfully. To see what this number was, I declared a new `a3index` and set it equal to the return value of the `a3trigInit` call. I ran the program and got to the breakpoint, then added a new Watch for this variable to see its value.



The Watch displayed the amount of values being returned was 10179, a *significantly* higher number than the `trigTable` array had size for. I assumed immediately that this was the issue: **the trig table didn't have enough room to store all of the needed values**, meaning the text could not properly initialize from these values because they simply did not completely exist.



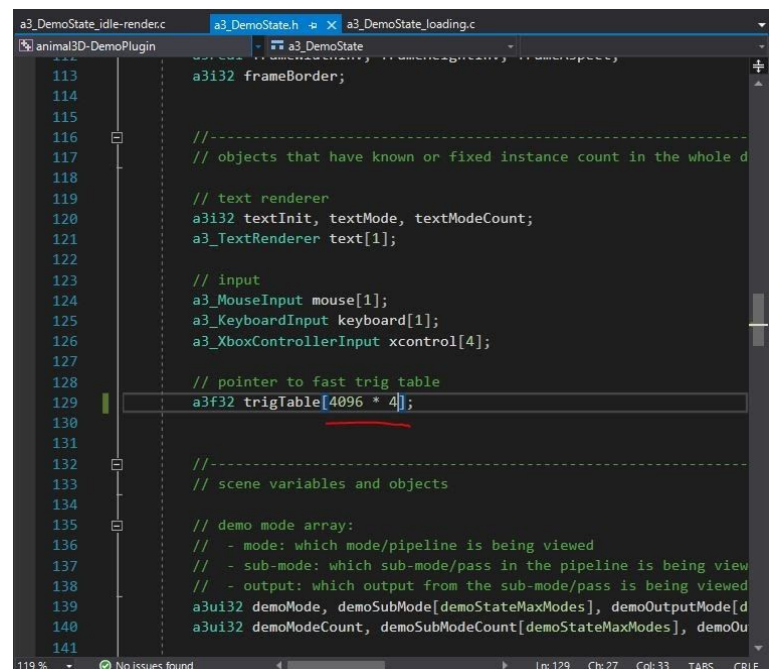
I opened several windows in Visual Studio and peeked the definition for `trigInit` to cross reference it with all of the data I had, and sure enough, it all made sense. The `trigTable` array is being set to only have 4096 elements when the actual number of values returned ($\text{samplesPerDegree} \times 720 + 1 \times 3$) is significantly higher, in this case 101079.

Solution

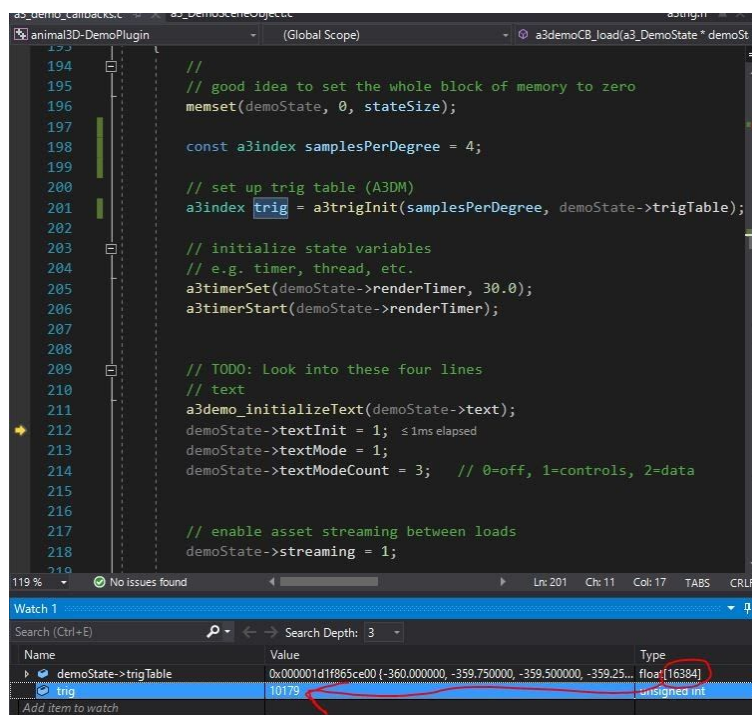
Based on the findings from the discovery and debugging process, I found the following solution to this bug:

Since the trigTable array held in the demoState was not large enough to hold all of the data it needed during initialization, the solution is to simply **increase the size of the trigTable array** to have enough space for all of them. I multiplied the provided number of 4096 by 4 to give enough space and some left over just in case

After applying this solution and running the program, we can see that the text is successfully being initialized since the program gets passed the call to initializeText without excepting. We can also see that the trigTable now has 16384 possibly elements, which is more than enough room for what we need to use with the table!



```
a3_DemoState_loading.c
113
114
115
116
117 //-----
118 // objects that have known or fixed instance count in the whole d
119
120 // text renderer
121 a3i32 textInit, textMode, textModeCount;
122 a3_TextRenderer text[1];
123
124 // input
125 a3_MouseInput mouse[1];
126 a3_KeyboardInput keyboard[1];
127 a3_XboxControllerInput xcontrol[4];
128
129 // pointer to fast trig table
130 a3f32 trigTable[4096 * 4];
131
132 //-----
133 // scene variables and objects
134
135 // demo mode array:
136 // - mode: which mode/pipeline is being viewed
137 // - sub-mode: which sub-mode/pass in the pipeline is being view
138 // - output: which output from the sub-mode/pass is being viewed
139 a3ui32 demoMode, demoSubMode[demoStateMaxModes], demoOutputMode[d
140 a3ui32 demoModeCount, demoSubModeCount[demoStateMaxModes], demoOu
141
```



```
a3demoCB_load(a3_DemoState * demoSt)
194
195 // good idea to set the whole block of memory to zero
196 memset(demoState, 0, stateSize);
197
198 const a3index samplesPerDegree = 4;
199
200 // set up trig table (A3DM)
201 a3index trig = a3trigInit(samplesPerDegree, demoState->trigTable);
202
203 // initialize state variables
204 // e.g. timer, thread, etc.
205 a3timerSet(demoState->renderTimer, 30.0);
206 a3timerStart(demoState->renderTimer);
207
208 // TODO: Look into these four lines
209 // text
210 a3demo_initializeText(demoState->text);
211 demoState->textInit = 1; // 1ms elapsed
212 demoState->textMode = 1;
213 demoState->textModeCount = 3; // 0=off, 1=controls, 2=data
214
215 // enable asset streaming between loads
216 demoState->streaming = 1;
217
218
219
```

Name	Value	Type
demoState->trigTable	0x000001d1f865ce00 (-360.000000, -359.750000, -359.500000, -359.25...	float[16384]
trig	10179	unsigned int

Bug 2: Geometry/Vertex Format Mismatch

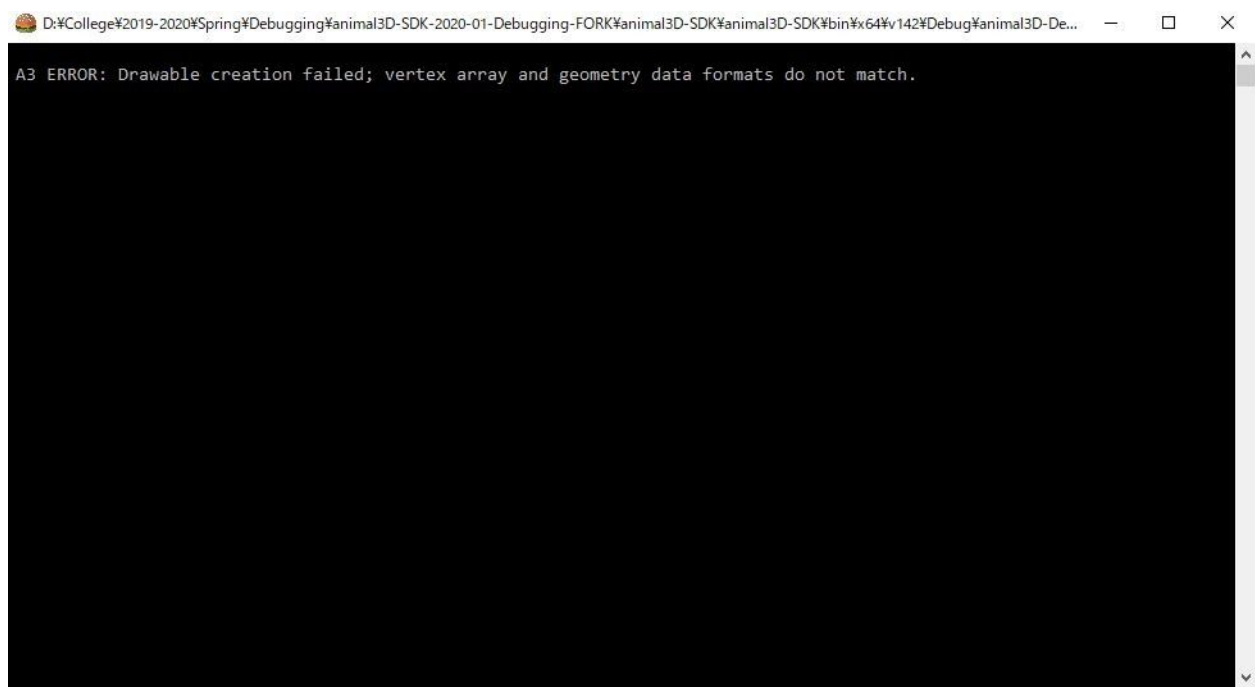
Description of Issue/Scenario

The core issue with the second discovered bug is a geometry loading error displayed in the console upon loading the demo state which says:

“A3 ERROR: Drawable creation failed; vertex array and geometry data formats do not match”

Details of Issue/Scenario

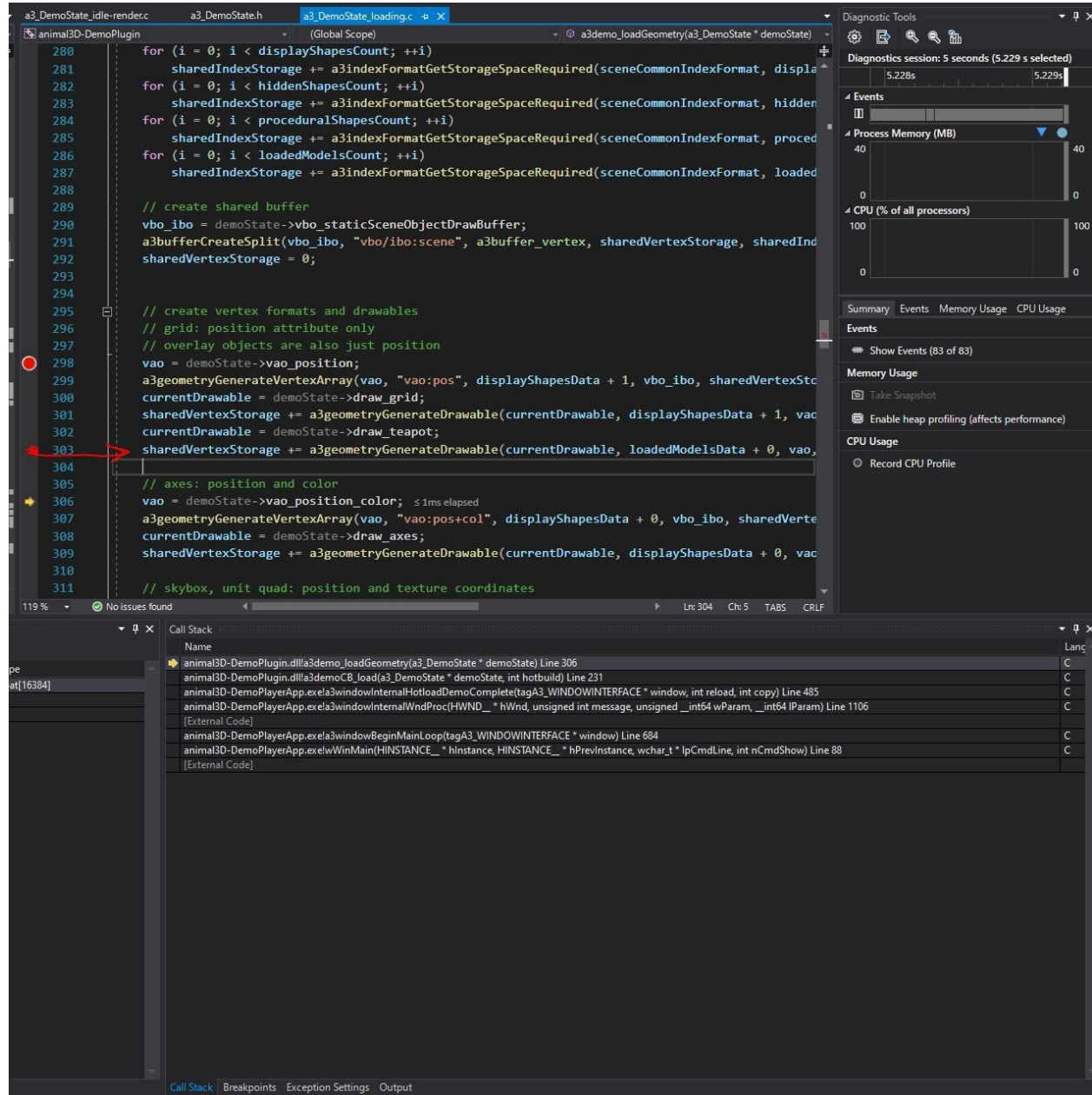
Like the first issue, this bug pertains to the initialization of the demo state, specifically the geometry objects that are loaded into the scene when it is successfully initialized. What’s interesting to note is that **no exception is actually thrown** when this happens; the a3D window remains blank. However, the console window displays an error message that is (thankfully) very descriptive of the issue:



This error message points in the exact direction for fixing this bug; it is somehow related to the vertex arrays and the geometry data that uses these arrays not having the same format, leading to the geometry not loading properly. What should be happening here is that the geometry all loads properly and the error message does not display, but somewhere during the loading process, a format mismatch

is occurring. After the error message is displayed, the a3D window continues to draw nothing and the program never crashes, since it is assumed that something is not loading properly and causing the program to run without ever displaying anything.

The issue was traced to line 303 in `a3_DemoState_loading.c`. After this line executes, the console displays the error message.



We can see that this is where the vertex arrays and geometry objects are being created and drawn to the screen. What's interesting to note is that the **grid draws successfully, but the teapot at line 303 does not**. This indicates that **not all of the objects are causing the problem, but the teapot specifically is**. Something is going wrong between the declaration of the vertex array and the teapot's geometry generation!

Discovery/Replication of Issue

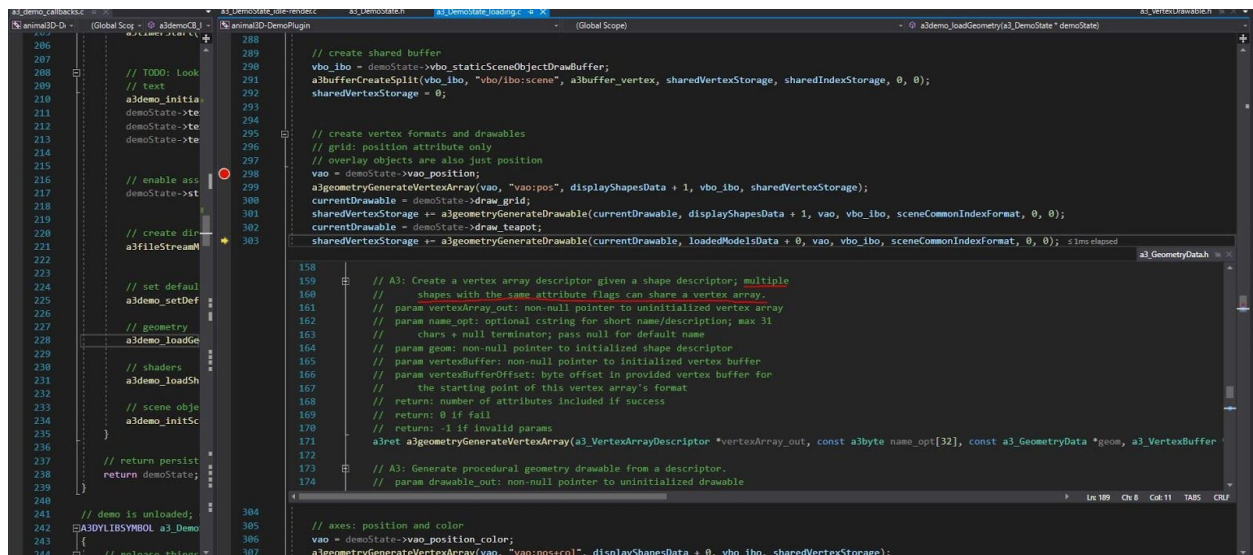
This issue was discovered immediately after the solution to Bug 1 was implemented by running the program again once built. To replicate, first implement the solution to Bug 1, then build and run the program again. Next, select “DEBUG: Demo project hot build and load -> Load without building”. Observe as the program loads but doesn’t display anything in the a3D window. Then observe the error message displayed in the console window.

To trace the issue to its root, place a breakpoint on a3_DemoState_callbacks.c line 228 to step into the loadGeometry function. Step through/over the various other calls until the error is displayed in the console window. This should occur after executing line 303 of a3_DemoState_loading.c.

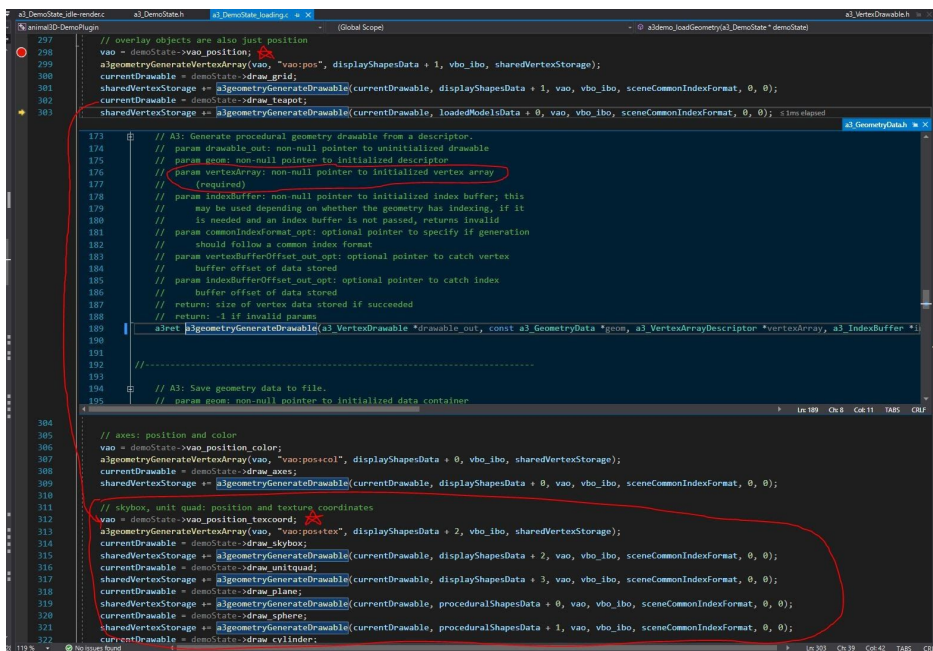
Debugging Process

To debug this issue, I used the same techniques and processes that I used to figure out why Bug 1 was happening: utilization of Watches on the changing variables (mainly the vertex array and sharedVertexStorage), using test variables to see the return value of the called functions, and reading the documentation for each function to cross-reference what was happening with what was supposed to opening.

The first thing I did was peek the definition for a3geometryGenerateVertexArray to see how it was supposed to operate since this is the first function called during the geometry loading process. What I found interesting was the phrase about how **multiple shapes with the same attributes could share vertex arrays**. This immediately clued me in as to what was happening; **a specific flag set by the vertex array wasn’t matching up to what the teapot object needed**, causing the format mismatch



With this in mind, I looked at the similarities and differences between the initialization of the teapot versus the other geometry objects (torus etc) and also peeked the definition for the `a3geomtryGenerateDrawable` function to see how that worked in relation to the vertex arrays. I found that the third parameter of this function (which was being passed the initialized vertex array variable “vao” for all of these calls) was a pointer to the initialized vertex array; in short, **vao is the variable that is causing the issue, as the flags being set for it do not match what the teapot needs!** Looking at the differences between the teapot generation and the other shapes, we can clearly see the problem: for the grid and teapot, **the only flag being set is for position**, whereas the shapes need **position and texcoords** to properly exist!



To confirm that this was the case, I created a test variable and set it equal to the teapot's generate function to see what it was returning. In the function definition, it is noted that a return value of -1 means that the function has invalid parameters. Sure enough, when I added a Watch to this test variable and executed the teapot creation, the test variable returned a value of -1. This

confirmed that the cause of this issue is because the teapot is being initialized in the wrong spot; it needs to be with the other texcoord shapes further down in the function call!

```
206 // TODO: Look...
207 // text
208 a3demo_initia
209 demoState->te
210 demoState->te
211 demoState->te
212 // enable ass
213 demoState->st
214 // create dir
215 a3fileStreamM
216
217
218
219
220
221
222
223
```

```
298 vao = demoState->vao_position;
299 a3geometryGenerateVertexArray(vao, "vao:pos", displayShapesData + 1, vbo_
300 currentDrawable = demoState->draw_grid;
301 sharedVertexStorage += a3geometryGenerateDrawable(currentDrawable, displa
302 currentDrawable = demoState->draw_teapot;
303 a3ret test = a3geometryGenerateDrawable(currentDrawable, loadedModelsData
304 sharedVertexStorage += test; ≤ 1ms elapsed
```

185 // param indexBufferOffset_out_opt: optional pointer to catch
186 // buffer offset of data stored
187 // return: size of vertex data stored if succeeded
188 // return: -1 if invalid params
189 a3ret a3geometryGenerateDrawable(a3_VertexDrawable *drawable o
190
191
192
193
194 // A3: Save geometry data to file.
195 // param geom: non-null pointer to initialized data container

Watch 1

Search (Ctrl+E) Search Depth: 3

Name	Value	Type
demoState->trigTable	0x000001c6789f0f80 [-360.000000, -359.750000, -359.500000, -359.250000, -359.000000, -358.750000, ...]	float[16384]
currentDrawable	0x000001c678a01b78 (first=0 count=0 primitive=0 ...)	a3_VertexDrawable *
test	-1	int

Solution

Once the issue was confirmed, the solution was a no-brainer: simply move “currentDrawable = demoState->draw_teapot” (line 302) and “sharedVertexStorage += a3geometryGenerateDrawable(currentDrawable...)” (line 303) **below the calls for drawing the torus**, or at line 326! This confirms that the bug is solved because the test variable then successfully returns the size of the vertex data!

```
325 // MOVED TEAPOT DOWN HERE SINCE IT NEEDS TEXCOORD!
326 currentDrawable = demoState->draw_teapot;
327 a3ret test = a3geometryGenerateDrawable(currentDrawable, loadedModelsData
328
329
330
331
332
```

182 // should follow a common index format
183 // param vertexBufferOffset_out_opt: optional pointer to catch
184 // buffer offset of data stored
185 // param indexBufferOffset_out_opt: optional pointer to catch
186 // buffer offset of data stored
187 // return: size of vertex data stored if succeeded
188 // return: -1 if invalid params
189 a3ret a3geometryGenerateDrawable(a3_VertexDrawable *drawable ou
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223

sharedVertexStorage += test;

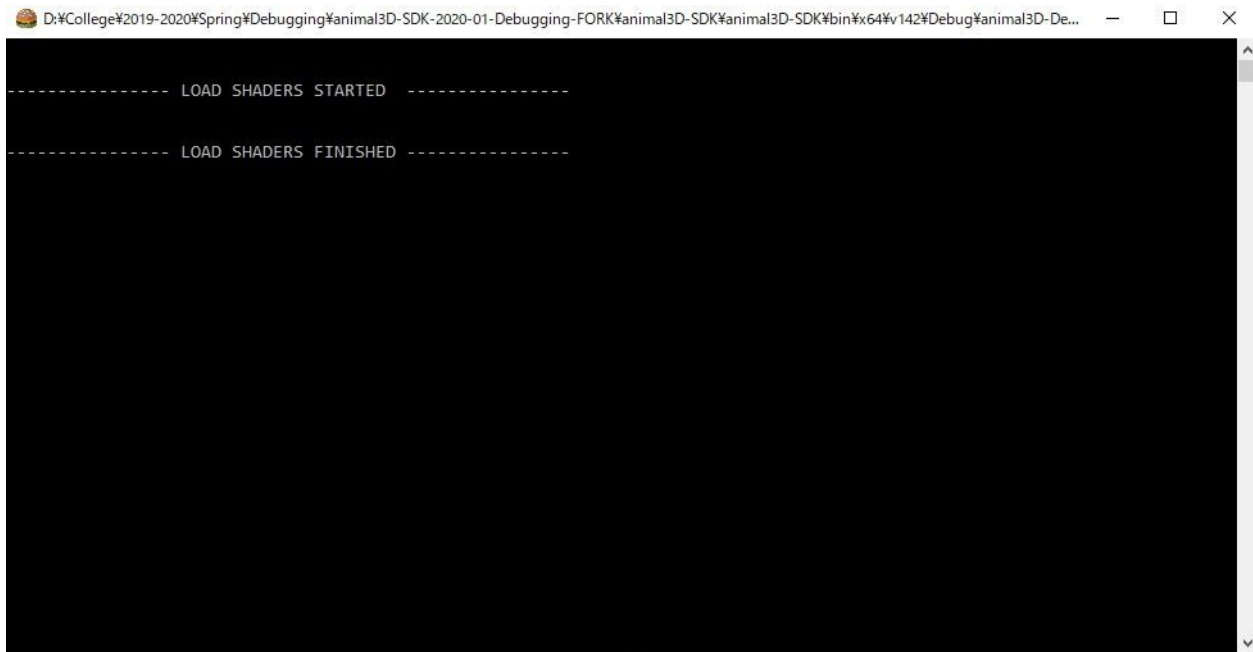
// release data when done
for (i = 0; i < displayShapesCount; ++i) ≤ 1ms elapsed

Watch 1

Search (Ctrl+E) Search Depth: 3

Name	Value	Type
demoState->trigTable	0x00000209297fb5a8 [-360.000000, -359.750000, -359.500000, -359.250000, -359.000000, -358.750000, ...]	float[16384]
currentDrawable	0x00000209297fb5a8 (first=0 count=47112 primitive=4 ...)	a3_VertexDrawable *
test	166680	int

Running the program after building with this solution removes the error from the console. However, the a3D window is still not drawing anything, meaning more bugs are at hand...



The screenshot shows a console window with a black background and white text. The text indicates that the shader loading process has started and finished successfully. The window title bar shows a file path: "D:\College\2019-2020\Spring\Debugging\animal3D-SDK-2020-01-Debugging-FORK\animal3D-SDK\bin\x64\Debug\animal3D-De...".

```
----- LOAD SHADERS STARTED -----  
----- LOAD SHADERS FINISHED -----
```

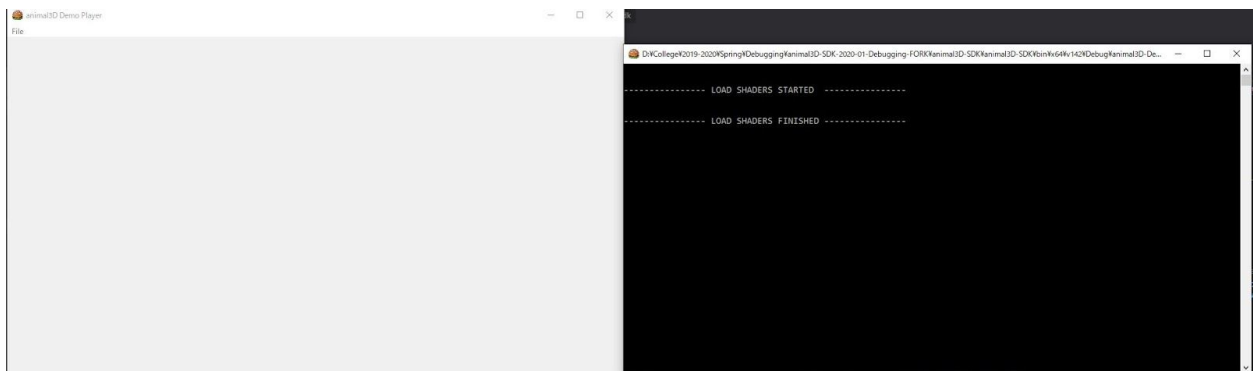

Bug 3: Refusing to Render

Description of Issues/Scenarios

The issue with this bug is that the demoState does not render anything when entering the idle window state; nothing is drawn to the screen and the a3D window remains blank

Details of Issues/Scenarios

This is a rather straightforward issue; even though the demo state is being loaded successfully according to console output, nothing is being drawn to the screen at all. The user cannot see the viewport and cannot move around a camera to see the various objects.



What should be happening here is that the viewport should show the demo state correctly and allow the user to move the camera around, but instead a blank screen continuously loops and does nothing. The program will run like this infinitely until it is closed. No exceptions are ever thrown, making the issue a **purely logical error**.

Discovery/Replication of Issues

Since this is a very straightforward and easily noticeable issue, it was discovered immediately after fixing Bug 1. It appears to be independent of Bug 2, since fixing Bug 2 did not resolve the issue. The bug was discovered to be unrelated to Bug 2 after building and selecting “DEBUG: Demo project hot build and load -> Load without building” again and observing that although the vertex/geometry error was gone from the console and the shaders were loaded, there was still nothing happening in the window.

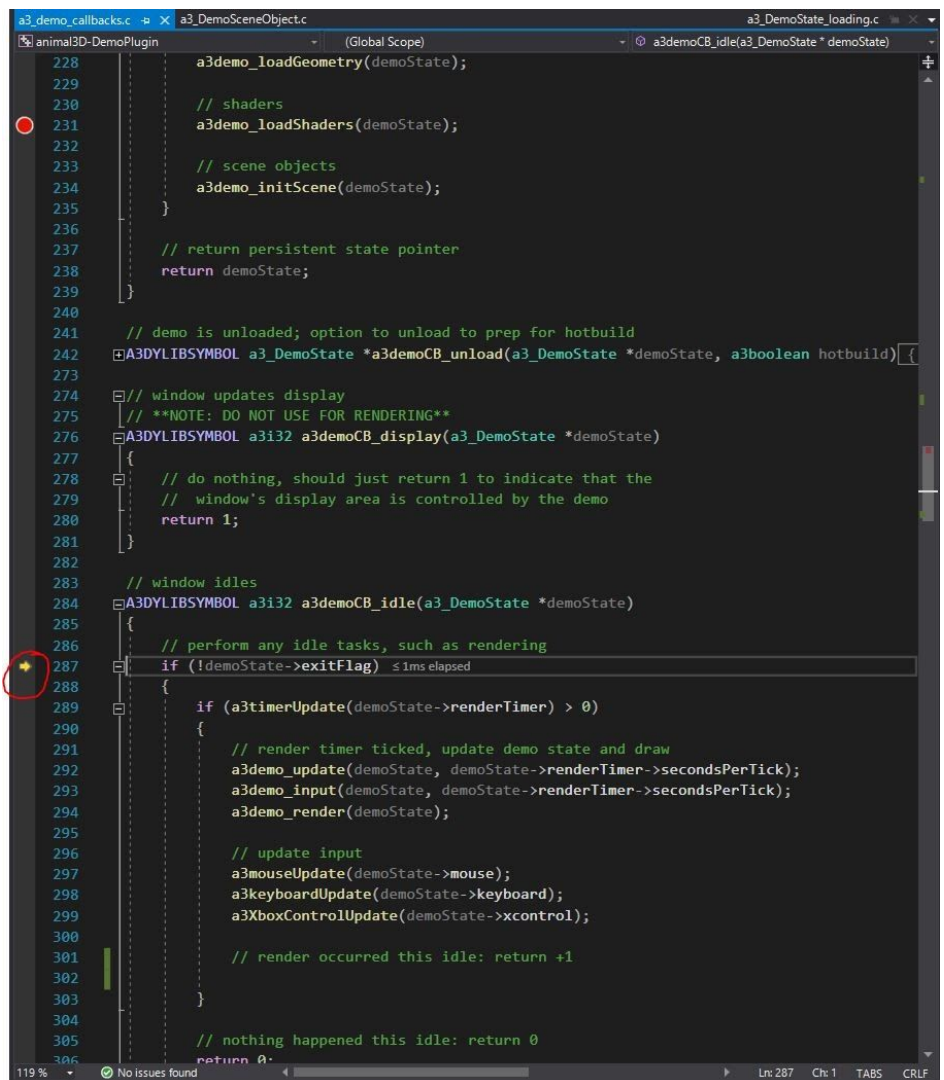
To replicate the issue, solve at least Bug 1 then build and run the program again. Next, select “DEBUG: Demo project hot build and load -> Load without building”. Observe as the program loads but doesn’t display anything in the a3D window, noticing how the shaders successfully load but nothing happens.

Wait a while to see if an exception will be thrown or the program will crash (hint: it won't), then close the program.

Debugging Process

Debugging this issue to find out why it was happening involved the use of only two techniques; using breakpoints and the call stack to trace where the bug was happening and then using the CTRL+F search function to find where exactly the program was performing render options.

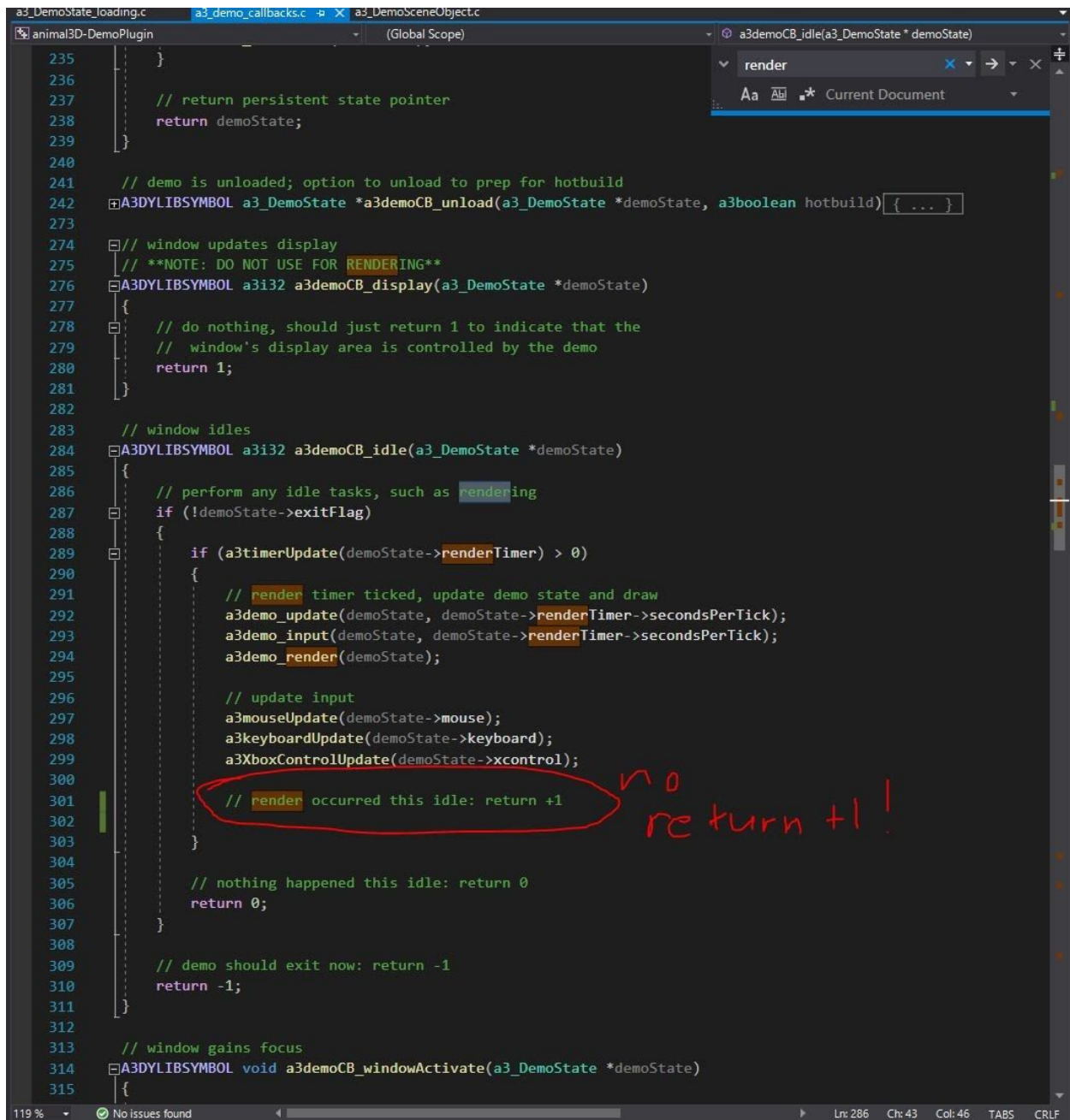
My first instinct was to place a breakpoint on the `a3demo_loadShaders` call in the demo state init function since the shaders were being successfully loaded; it was a good starting point to see where things were going wrong. I then manually stepped into every call afterwards to see where the program was going to start rendering things. Since the provided code has no access to the main window loops, it ended up being a dead end for a while (the program would get to the loop and then proceed as normal since there was no real "error") until eventually I got it to step inside the `a3demo_CB_idle` function. I could see from this function that this is where the render loop was happening on idle, so I decided to look through it to see if the issue was here.



```
a3_demo_callbacks.c x a3_DemoSceneObject.c a3_DemoState_loading.c
animal3D-DemoPlugin (Global Scope) a3demoCB_idle(a3_DemoState *demoState)
228 a3demo_loadGeometry(demoState);
229
230 // shaders
231 a3demo_loadShaders(demoState);
232
233 // scene objects
234 a3demo_initScene(demoState);
235
236 }
237 // return persistent state pointer
238 return demoState;
239
240
241 // demo is unloaded; option to unload to prep for hotbuild
242 A3DYLBSYMBOL a3_DemoState *a3demoCB_unload(a3_DemoState *demoState, a3boolean hotbuild){
243
244 // window updates display
245 // **NOTE: DO NOT USE FOR RENDERING**
246 A3DYLBSYMBOL a3i32 a3demoCB_display(a3_DemoState *demoState)
247 {
248 // do nothing, should just return 1 to indicate that the
249 // window's display area is controlled by the demo
250 return 1;
251 }
252
253 // window idles
254 A3DYLBSYMBOL a3i32 a3demoCB_idle(a3_DemoState *demoState)
255 {
256 // perform any idle tasks, such as rendering
257 if (!demoState->exitFlag) & 1ms elapsed
258 {
259 if (a3timerUpdate(demoState->renderTimer) > 0)
260 {
261 // render timer ticked, update demo state and draw
262 a3demo_update(demoState, demoState->renderTimer->secondsPerTick);
263 a3demo_input(demoState, demoState->renderTimer->secondsPerTick);
264 a3demo_render(demoState);
265
266 // update input
267 a3mouseUpdate(demoState->mouse);
268 a3keyboardUpdate(demoState->keyboard);
269 a3XboxControlUpdate(demoState->xcontrol);
270
271 // render occurred this idle: return +1
272
273 }
274
275 // nothing happened this idle: return 0
276 return 0;
277 }
```

Sure enough, I was onto something; I used the Find function with CTRL+F to search for any references to the word "render" and followed the trace until I got to line 301 of `demoCallbacks`. This line has a comment saying "render occurred this idle: return +1". There are other comments at the end of the other code paths in this function that state the idle state returns: a value of 0 means nothing happened; a

value of -1 exits the demo. So where's the +1? Well...**that's the bug!** It was a purely logical error; the flag for a successful render was never being returned, so the screen never updated anything!



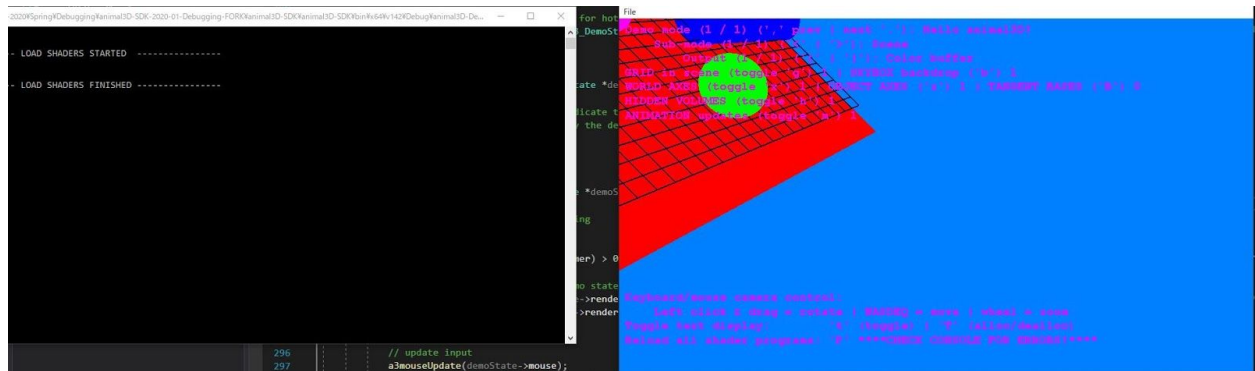
```
235 }
236
237 // return persistent state pointer
238 return demoState;
239 }
240
241 // demo is unloaded; option to unload to prep for hotbuild
242 A3DYLIBSYMBOL a3_DemoState *a3demoCB_unload(a3_DemoState *demoState, a3boolean hotbuild) { ... }
243
244 // window updates display
245 // **NOTE: DO NOT USE FOR RENDERING**
246 A3DYLIBSYMBOL a3i32 a3demoCB_display(a3_DemoState *demoState)
247 {
248     // do nothing, should just return 1 to indicate that the
249     // window's display area is controlled by the demo
250     return 1;
251 }
252
253 // window idles
254 A3DYLIBSYMBOL a3i32 a3demoCB_idle(a3_DemoState *demoState)
255 {
256     // perform any idle tasks, such as rendering
257     if (!demoState->exitFlag)
258     {
259         if (a3timerUpdate(demoState->renderTimer) > 0)
260         {
261             // render timer ticked, update demo state and draw
262             a3demo_update(demoState, demoState->renderTimer->secondsPerTick);
263             a3demo_input(demoState, demoState->renderTimer->secondsPerTick);
264             a3demo_render(demoState);
265
266             // update input
267             a3mouseUpdate(demoState->mouse);
268             a3keyboardUpdate(demoState->keyboard);
269             a3XboxControlUpdate(demoState->xcontrol);
270
271             // render occurred this idle: return +1
272         }
273
274         // nothing happened this idle: return 0
275         return 0;
276     }
277
278     // demo should exit now: return -1
279     return -1;
280 }
281
282 // window gains focus
283 A3DYLIBSYMBOL void a3demoCB_windowActivate(a3_DemoState *demoState)
284 {
285 }
```

no return +1!

Solution

Once the source of the issue was confirmed, it was instantly solvable; the comment at line 301 of demoCallbacks literally spells out the answer! Simply add “return +1” at line 302 of demoCallbacks, then build and run the program and select “DEBUG: Demo project hot build and load -> Load without building”. That’s it!

```
284  A3DYLIBSYMBOL a3i32 a3demoCB_idle(a3_DemoState *demoState)
285  {
286      // perform any idle tasks, such as rendering
287      if (!demoState->exitFlag)
288      {
289          if (a3timerUpdate(demoState->renderTimer) > 0)
290          {
291              // render timer ticked, update demo state and draw
292              a3demo_update(demoState, demoState->renderTimer->secondsPerTick);
293              a3demo_input(demoState, demoState->renderTimer->secondsPerTick);
294              a3demo_render(demoState);
295
296              // update input
297              a3mouseUpdate(demoState->mouse);
298              a3keyboardUpdate(demoState->keyboard);
299              a3XboxControlUpdate(demoState->xcontrol);
300
301              // render occurred this idle: return +1
302              return +1;
303          }
304      }
305  }
```



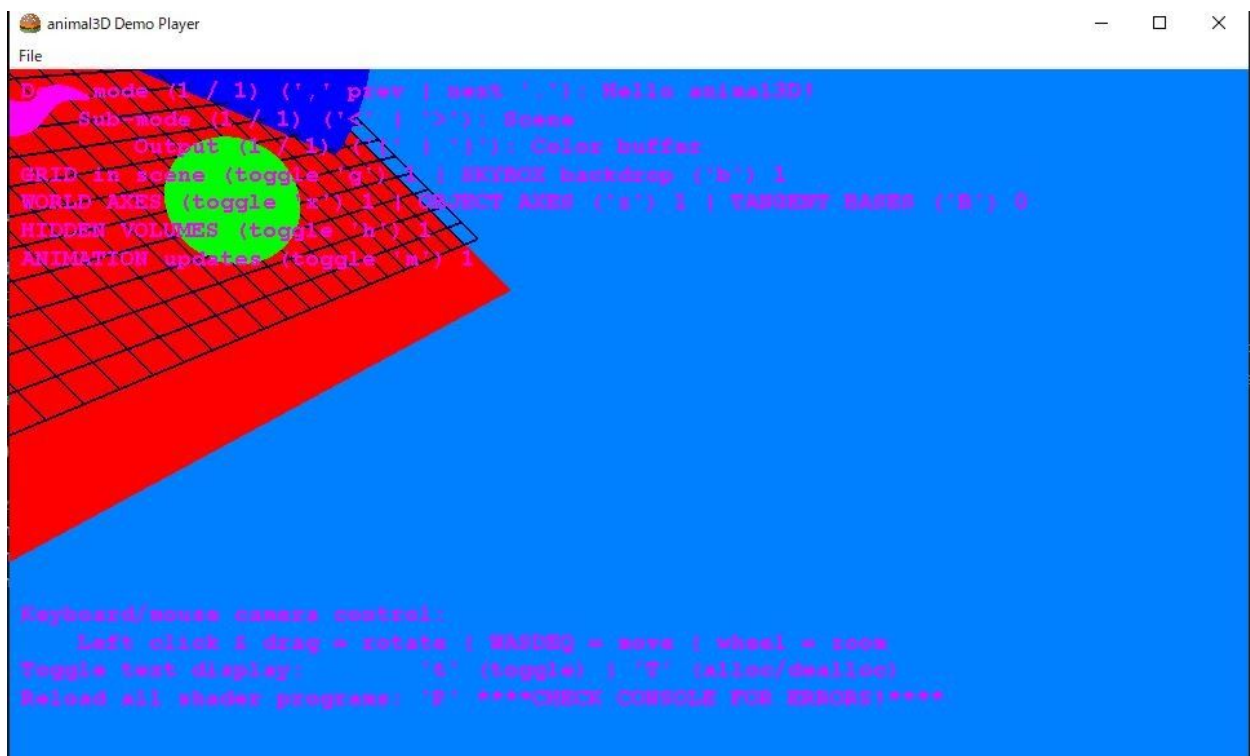
Bug 4: Rotation Frustration

Description of Issues/Scenarios

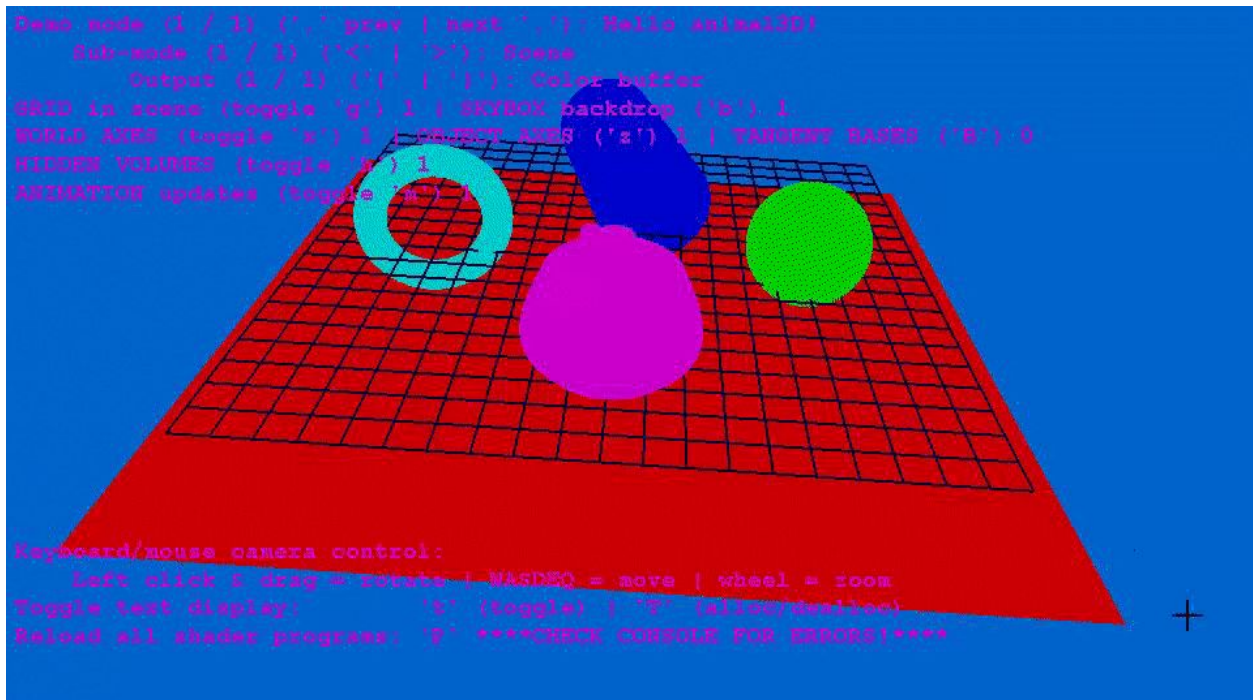
This issue pertains to a problem with the rotation math for the scene camera. When initialized, the camera is rotated in the wrong position from the targeted default and it moves on the **roll axis instead of the yaw axis** with left and right mouse movement.

Details of Issue/Scenario

When the scene is loaded, the camera is pointed in a strange position that can't really see the scene. What should happen is that the camera should be at a specific angle where the demo objects can be seen immediately without having to move it around.



When the user uses the left mouse button to rotate the camera, moving it left and right causes the camera to move on the **roll axis**. What should happen is that the camera should move on the **yaw axis** as is standard for cameras in game engines. The pitch axis appears to be working correctly. Here's an *image of me awkwardly trying to move the camera to a good position with these funky controls
this is a gif; if it doesn't play, check NOTES/Images/Bug 4/ 2. Rotation not working gif.gif)



Discovery/Replication of Issues

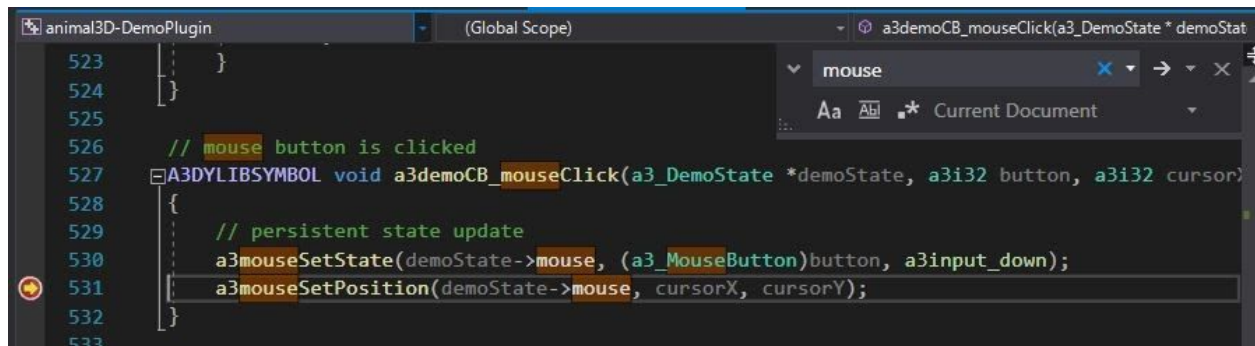
This issue was discovered immediately after solving Bug 3 and trying to use the camera. To replicate, fix all three previous bugs then build and run the program. Select "DEBUG: Demo project hot build and load -> Load without building" from the drop down like always and note how the camera is weirdly placed in the corner somewhere. To see how the controls are broken, hold left click and move the mouse left and right; the camera will roll like an airplane instead of panning left and right. Note how the pitch axis feels correct.

Debugging Process

The process for debugging this issue to figure out why it is happening involved the use of the CTRL+F search tool as well as breakpoints and Watches to see the variables controlling the issue.

To first find out where things were happening, I did a CTRL+F search for "mouse" to see if I could find where the mouse data was being sent to the camera for rotations. I found a call for mouseClicked in demoStateCallbacks and placed a breakpoint there. This would let me follow the mouse when input was received to see where it goes to the camera. This turned out to be a dead end; stepping into everything

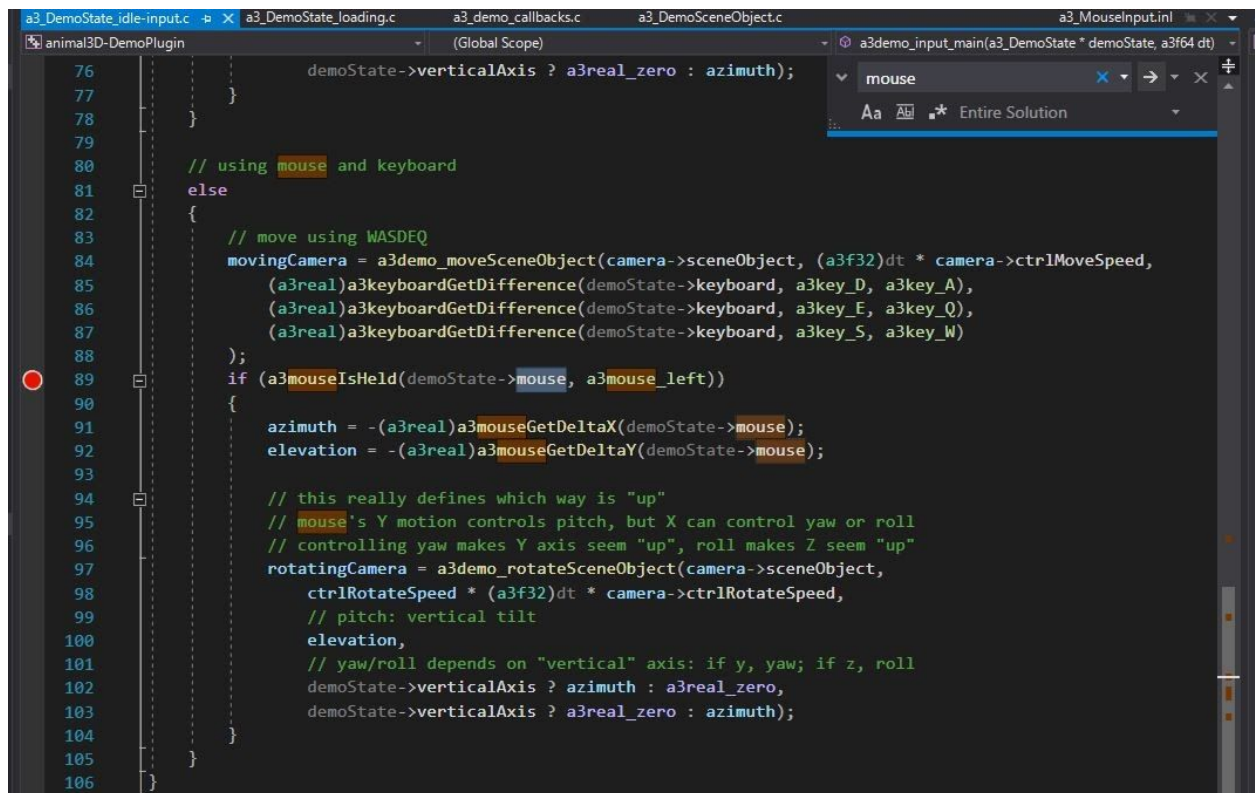
eventually led to the program going back to the loop since no actual exception error was occurring.



The screenshot shows a code editor with a file named 'animal3D-DemoPlugin'. The code is in C and defines a function 'a3demoCB_mouseClick' which is called when a mouse button is clicked. The function updates the mouse state and position in the 'demoState' structure. The code is as follows:

```
523 }
524 }
525
526 // mouse button is clicked
527 A3DYLIBSYMBOL void a3demoCB_mouseClick(a3_DemoState *demoState, a3i32 button, a3i32 cursor)
528 {
529     // persistent state update
530     a3mouseSetState(demoState->mouse, (a3_MouseButton)button, a3input_down);
531     a3mouseSetPosition(demoState->mouse, cursorX, cursorY);
532 }
533
```

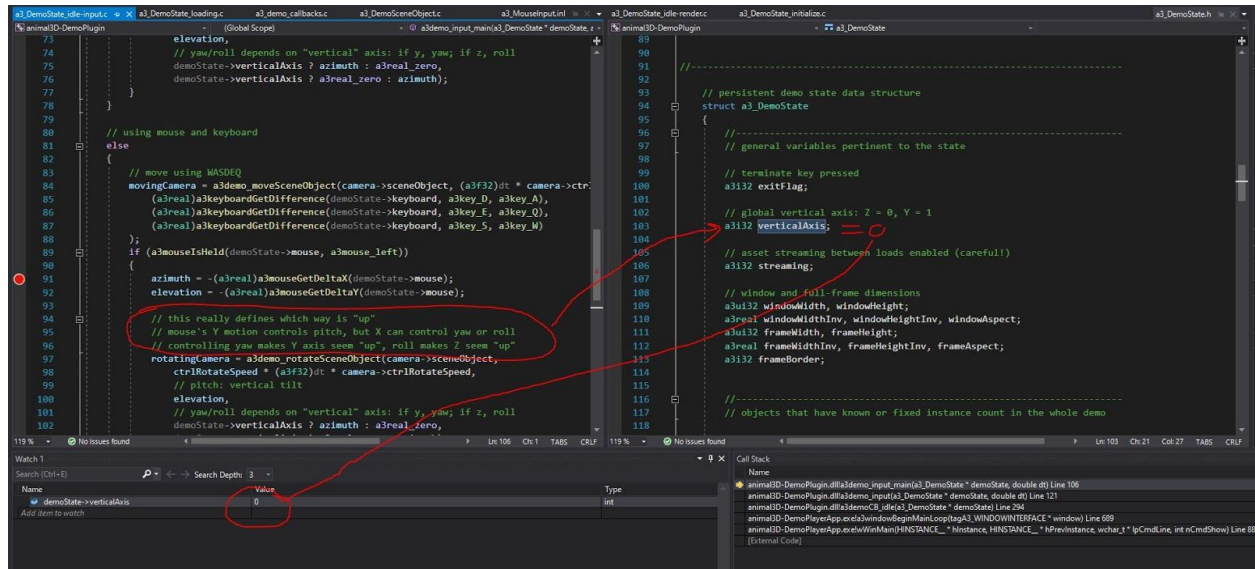
Since I couldn't find where the mouse input was going to the camera in callbacks, I decided to try searching the entire project for "mouse". I searched through every mention of mouse (boy are there alot) until I found what I was looking for in a3_DemoState_idle-input.c. Around line 85 in this file is where the input is happening for the mouse when the camera is moved around. The comment block at line 94 gives a huge clue to what's happening here; the vertical axis of the demoState determines whether X mouse movements control yaw or roll!



The screenshot shows a code editor with a file named 'a3_DemoState_idle-input.c'. The code is in C and handles mouse input for camera movement. The code is as follows:

```
76 demoState->verticalAxis ? a3real_zero : azimuth);
77 }
78 }
79
80 // using mouse and keyboard
81 else
82 {
83     // move using WASDEQ
84     movingCamera = a3demo_moveSceneObject(camera->sceneObject, (a3f32)dt * camera->ctrlMoveSpeed,
85     (a3real)a3keyboardGetDifference(demoState->keyboard, a3key_D, a3key_A),
86     (a3real)a3keyboardGetDifference(demoState->keyboard, a3key_E, a3key_Q),
87     (a3real)a3keyboardGetDifference(demoState->keyboard, a3key_S, a3key_W)
88 );
89 if (a3mouseIsHeld(demoState->mouse, a3mouse_left))
90 {
91     azimuth = -(a3real)a3mouseGetDeltaX(demoState->mouse);
92     elevation = -(a3real)a3mouseGetDeltaY(demoState->mouse);
93
94     // this really defines which way is "up"
95     // mouse's Y motion controls pitch, but X can control yaw or roll
96     // controlling yaw makes Y axis seem "up", roll makes Z seem "up"
97     rotatingCamera = a3demo_rotateSceneObject(camera->sceneObject,
98     ctrlRotateSpeed * (a3f32)dt * camera->ctrlRotateSpeed,
99     // pitch: vertical tilt
100     elevation,
101     // yaw/roll depends on "vertical" axis: if y, yaw; if z, roll
102     demoState->verticalAxis ? azimuth : a3real_zero,
103     demoState->verticalAxis ? a3real_zero : azimuth);
104 }
105 }
106
```

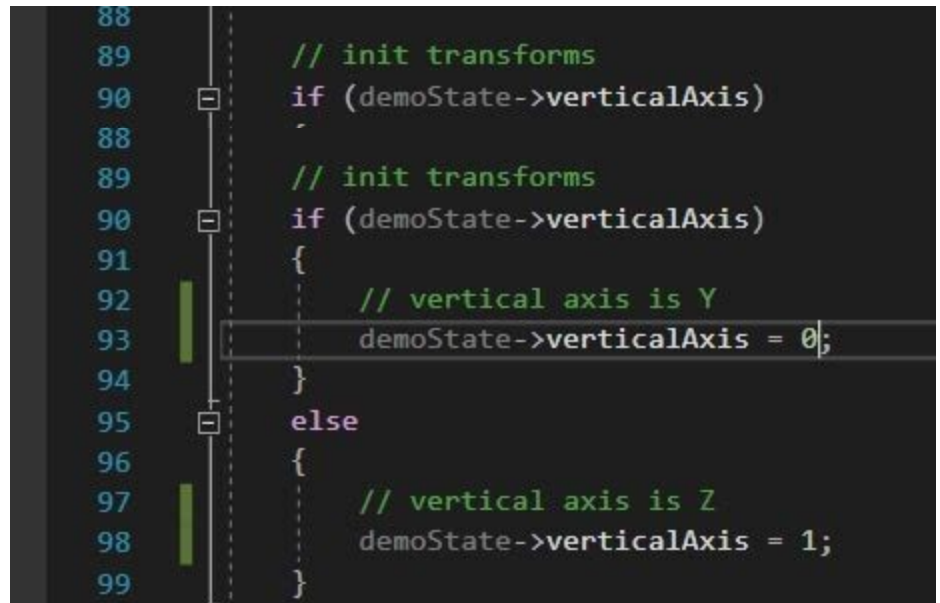
I placed a breakpoint here and put a Watch on `demoState->verticalAxis` to see its value. Sure enough, it was 0, which according to the header, means it is using the roll axis instead of the yaw axis.



Solution

I tried to fix the issue by simply setting `verticalAxis` to 1 in the header, but for some reason the compiler refused to tell me, it didn't work. I attempted fixing the issue inside of `a3_DemoState_initialize` at line

93 and 98; this is where the transforms are initialized. I tried swapping them here, which did in fact fix the axis issue, but...



...now all the axis are messed up. Moving left and right is now on the yaw axis, but its still adding pitch input to the screen. Using mouse y inputs now causes the camera to roll, meaning there is another bug causing this to take place...

