I'm not robot
reCAPTCHA
Continue

3d game maker studio 2 download

I will leave (dd/mm/y) formatted date here, so you know when it was last updated: 16/04/18 - as IDE: 2.1.4.288, RUNTIME: 2.1.4.203 - Added a tiny tidbit on layer_force_draw_depth()ie setup section I'll also update the guide if there are better methods that I haven't thought about/learned yet. I'M NOT UPDATING THIS PAGE ANYMORE. ADDITIONAL UPDATES OF THIS GUIDE CAN BE FOUND ON MY WEBSITE UPDATED GUIDE HERE (Includes a bit of info on transparency and rendering order): website offers full syntax highlighting and documentation links also to make guide-writing much easier. I've received/seen some questions about getting 3D working, and I want to get involved in the forum now that GMS2 is coming, so here's a kickstarting guide. I have posted this guide to the new GMS2 discussion board because it's not really appropriate to post it on another forum yet. Once a special forum for GMS2 programming + guides is created, I will request that this guide be moved there. I recommend you @YellowAfterlife plugin for highlighting the forum code syntax. This makes the reading code here way easier: GM version: IDE: 2.1.4.288, RUNTIME: 2.1.4.203 Target Platform: ALL Download: Built Project (Zip, so free users can have a search!) Links: N / Summary: So, do you want to start 3D with GMS2? Well, here are a few things you should know first: I will be referring to GameMaker Studio 2 as GMS2 throughout the guide because I'm too lazy to keep writing it in full. This guide is written with beginners in 3D mind. If I'm explaining something and you think you already know all about it, just skip it. Alternatively, if something I have written is confusing, do not be afraid to ask, and I will try to be clearer! I will reference the matrix/matrix a lot. They are necessary to do almost anything in 3D, but you don't necessarily need to know how they work. A lot of the matrix is autogenerated and assigned code, never need to know how they work. However, knowing how they work can help a lot. GMS2, similar to GM: S 1.X, no 3D engine. It is mainly designed for 2D, so 3D games require a lot of manual work to build. You need to know how to use GML. I don't know if any D& D features have been added to 3D (I don't think so anyway), but this guide will be GML based anyway. This guide may be easier if you ever use d3d_* functions, but they no longer exist. This guide is written with beginners in 3D mind anyway. You can no longer import models using d3d_model_load (well, you can, but that uses compatibility scripts and is not recommended), so it is recommended to write your own model importer/exporter. I'm going to have to go back to the 1997-03-10.2009. The 100 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 00 00 00 00 00 00 00 0 &buffer_load I'm not covering HUD up because it works just like it has been since 1.X (draw GUI) and already has guides and help available. Remember to use the guide! It answers many questions that you might have and can give you more information about the feature. If the guide does not help you, you can ask a question here and I will try to help! Now that's out of the way, here's a list of what I'll be covering: Create 3D requirements and 3D projection with the new camera Using a vertex buffer to create a pattern and make it transform models with the new matrix functionality Other useful d3d_* features, which has been replaced (texture repeat, texture interpolation, backface melting, etc.) Other odds-n-ends I think I should consider (batching, layer depth) I also cover some basic shader use in the future (both GLSL ES and HLSL), but only when a full release comes because we currently don't have access to shaders. I also cover using lighting, but they are more useful when combined with shaders so I leave the lights up for a full release too. UPDATE ON THIS ^^: I don't think I really cover any shader stuff in this special tutorial – it's a great topic that covers both 2D and 3D, with loads of guides out already. I could take a more 3D-driven guide, or I could just release some extensions with 3D shaders, with a bunch of commented code to help you get an idea of what's going on. Lighting is an intersing one though. Hmm. If you go through the guide from start to finish, it's pretty much what you'll see: Not super impressive, but it applies the basics. Relevant documentation links: Camera Vertex buffers Matrices So lets start then! Tutorial: 1) Creating 3D requirements and 3D projection with the new camera To start with the following code, all go to create a case of some kind of control object, unless I specify otherwise. Well, before we even get started, we need to make a few things. In 3D, we have something called a z-buffer – it basically allows sorting to make arrangements so that things that are further away seem to be behind things that are closer (note that this behavior can change GMS2, . If we do not enable z buffer, then rendering will look very strange, because things that are far away can be done on top of things that are closer (even within one model!) To enable the z buffer, we simply use this code: gpu_set_zwriteenable(true);//Allow you to write z to the buffer gpu_set_ztestenable(true);//Allows in-depth testing, so far things are drawn, and you can also find useful to force layers to be done in depth from 0 (this will stop layers translation for what you do on any layer – although you may find the default behavior preferably in some cases) layer_force_draw_depth (true, 0); (documentation layer_force_draw_depth) Believe it or not, that's all it takes to get rendering in 3D! Now we can move on to the next step - create and use the camera in a 3D environment. Since the GMS2 update, opinions have been completely replaced by cameras, but it has d3d projection features – they are now the same one that is much more convenient when you get used to them. The first thing we want to do is create a camera, give it a view, and enable views and make your visible. You can still create an opinion room editor, but sometimes it's nice to have all the code in front of you. Technically speaking, you don't even need to create a 3D projection, however it makes managing the camera much easier. Note that as before, the size of the game window will be the size of the first room view port/room by default. For this reason, you must have a set-up space to get things ready and set the size to make it useful. Otherwise, you could use window_* functions together view_set_Xport to get the size you want. Anyway, code! First, getting our opinion actually appears (you don't need to do this if you've set the view room editor) // First, we need to provide an opinion and make our views visible (in this case, see 0) view_enabled = true;//Enable to use the view view_set_visible (0, true);//Make this view visible next, we need to create a camera and make it fully functional. To do this, we need to create a camera, assign a projection matrix and associate with the camera. We will keep the camera variable, although we will use view_camera [0] to refer to it later. We keep the camera variable if we want to associate the camera with another view later, perhaps in another room, without re-creating it. First, create a camera. We could use camera_create_view, but it's more useful for 2D environment camera = camera_create(); Then we need to build a projection matrix. I keep it, for example, in the scope if I need to grant it later. (Although you can download matrices from the camera with camera_get features//I use matrix_build_projection_perspective_fov because it gives the most control over how your predictions look. / / Here's how I use the arguments: I give a 60 degree vertical point of view, with a ratio of view_wport/view_hport, with 32 units off the clipping plane, and 3.2000 far from the cutting plane. Some of these values may need tweaking to your liking. projMat = matrix_build_projection_perspective_fov (-60, -view_get_wport (0)/view_get_hport (0), 32, 32000); Now we assign a projection matrix to the camera_set_proj_mat (camera, projectMat); Finally, we bind the camera with a view view_set_camera (0, camera); NOTE: GMS2, setting the projection differs from 1,4 - default, it's now this strange conversion of right hand coordinate space stay under 2D, but it makes the up-vector act strange. To projection act like 1.4 and before and use left-handed space and make an up-vector act as expected, you must make the fov and ratio values negative, hesed the code above. You may notice that we have assigned a projection matrix, but actually don't point the camera to anything. This happens later, and separately - this is one of the benefits of the new camera system - we only need to create a projection once, thus saving a lot of time compared to calling d3d_set_projection_ext each drawing frame, because half the math has already been done. If you want your camera to have an orthographic projection (not a perspective), replace the matrix compilation with this function: matrix_build_projection_ortho (view width, view height, znear, zfar); There is one more thing you need to do to get your camera ready! Note that we still haven't assigned a lookat matrix to the camera! It is used to actually point the camera from location to location. Now you could update this matrix step or draw, but the camera comes with a new, cool feature – camera_set_update_script() (has begin_script and end_script features too! See the guide to learn more about them!). Basically, this script is called each to make an update in one view, assuming that the camera is associated with the view. This means that you can enable frame skipping, or disable the view, and the camera won't waste time updating what you don't need! So, to use the camera update script, you must create a script. Ideally, the content of these scripts should be able to be autonomous, and only external references would be globals or case variables that you know exist, otherwise the game will probably crash. For this demo, I just created a camera that could spin around the center of the room based on time. The script is camera_update_script and contains code: //Set up camera location var zz = -640; var xx = lengthdir_x (720,-current_time/10) + (room_width* 0,5);//Rotation is now negative to match the old gif and spin clockwise var yy = lengthdir_y (720,-current_time/10) + (room_height* 0.5); Create a matrix that looks from the camera location above to the center of the room. Up vector points to -z mLookat = matrix_build_lookat (xx, yy, zz, (room_width * 0,5), (room_height * 0,5), 0, 0,0, -1); Assign a matrix to the camera. These updates were camera searches from, without needlessly updating the projection. camera_set_view_mat(view_camera[0], mLookat; Note that the reference [0] view_camera [0] is a reference. I have done this because it is the only right way to guarantee that we target a camera that owns a view0. As another point of interest, you could create a test to make camera updates lookat matrix only if it has moved or changed orientation, which can save even more processing time. Ok, the last thing you need to do is assign an update script Back create the event (if your other code code be), simply add the following: //Assigns an update script called camera_update_script to the camera that belongs to the view0000 camera_set_update_script (view_camera [0], camera_update_script); That's all! If you create this camera in a simple, tiled room, it will look something like this: Aim and moving your camera around using user input is a little more work and more project-specific (for example, a 3rd person will have different controls for the first person, and the flight sim would be different again), but as long as you have a good understanding of algebra and trigonometry, it would be pretty simple. 2) Using the apex buffer to create the pattern and make it OK compared to the camera, it is probably harder to understand. However! Don't get it! I kept putting off learning bumper-related things for years, just because the word buffer – it sounded hard and scary, but they're pretty simple – especially in the context of peak buffers, where a lot of work is done for you when it comes to reading the bumper. Again, we define most of this stuff to create an event unless otherwise stated. So, the apex buffer is used to create patterns – or perhaps more relevant – vertex layouts – they work in both 2D and 3D, and allow you to do things with stuff like custom attributes, or with the bare minimum data needed to make it to reduce overhead. When used in conjunction with shaders, they are super powerful. Apex buffer is effectively composed of 2 parts: - a format that determines what information the buffer contains - a buffer that actually contains the information transferred to shader rendering The first thing you want to do is build peaks in the format. The format contains information about what each buffer vertex will contain. The one we will create will be 3D position, color and texture coordinates. It is important to remember the order in which you define the information in the vertex format, because the data attached to the buffer must be in the same order. Each defined apex also has all the information, even if you feel a special apex doesn't need a part of it. NOTE: The format described below indicates what we lovingly call a full fat format – it has all the elements needed to use GameMakers' default shader – if you miss any of these elements, the drawing will have an error with an invalid input layout. If you want to use fewer components, you need to write shader. This may have some advantages, such as slightly lower memory usage and lower overhead when the buffer is sent to the shader. Code: //Begin defining the format vertex_format_begin(); vertex_format_add_position_3d ();/Add 3D position info vertex_format_add_color();/Add color info vertex_format_add_textcoord();/Texture coordinate info //End building the format, and assign the format to the variable format format = vertex_format_end(); Now we have a format we can create and build a vertex buffer. In this case , I building a simple, white plane plane flat type instead of flying one) built with a triangle list use in mind (to see more information about it, check the manual on draw_primitive_begin. You can create more complex models, but it would be a little time consuming for this tutorial. To build a plane, we actually need 2 triangles. They are defined as : //Create vertex buffer. Another function that vetex_create_buffer_ext can be used to create a buffer with its size above and fixed. With the vertex_create_buffer, the buffer will only automatically grow as needed. vb_plane = vertex_create_buffer(); Start buffering using the predefined vertex_begin (vb_plane, format); Using the size to keep it square, if we decide to change how bug it is. var size = 32; Add six peaks need to draw a simple square plane. The first triangle vertex_position_3d (vb_plane, -size, -size, 0); vertex_color. Article 10 vb_plane. c_white. Article 10(1) vertex_texcoord(vb_plane, 0.0); vertex_position_3d (vb_plane, size, -size, 0); vertex_color. Article 10 vb_plane. c_white. Article 10(1) vertex_texcoord(vb_plane, paragraph 1 and 0); vertex_position_3d (vb_plane, -size, size, 0); vertex_color. Article 10 vb_plane. c_white. Article 10(1) vertex_texcoord(vb_plane, 0.1); Second triangle. The winding order has been saved, so the drawing is consistent when culling is enabled. vertex_position_3d (vb_plane, -size, size, 0); vertex_color. Article 10 vb_plane. c_white. Article 10(1) vertex_texcoord(vb_plane, 0.1); vertex_position_3d (vb_plane, size, -size, 0); vertex_color. Article 10 vb_plane. c_white. Article 10(1) vertex_texcoord(vb_plane, paragraph 1 and 0); vertex_position_3d (vb_plane, size, size, 0); vertex_color. Article 10 vb_plane. c_white. Article 10(1) vertex_texcoord. Article vb_plane, (1) and (1); Complete buffering. vertex_end(vb_plane; All that's left is to make the apex buffer – really, that's it! I even set up an airplane to make texture coordinates correctly, so it can be a texture that is good for floors or ceilings. So, let's see the monster code that is required to make this buffer. It goes to draw an event://Arguments are vertex_buffer, primitive type and texture. vertex_submit(vb_plane, pr_trianglelist, -1); And that's how to build a basic apex buffer and make it. Hopefully you'll be able to use this info to build a buffer that's a little more complicated than a flat plane, which has little use anyway because the new tiles work in 3D. Another thing to note is that you want to delete your apex buffers and formats when you're done with them to free up memory. The vertex buffers are deleted with vertex_delete_buffer(buffer), and the formats are deleted with vertex_format_delete(format). Note that the format can only be deleted if all buffers that use it are deleted, otherwise the runner will crash. There is also vertex_freeze (buffer) function - it will disable buffer editing/updating, but performance. So, now you know how to create a apex buffer, but it can be a

painfully tedious process. It's good to understand who they are and how they they (and really useful for creating vibrant eyes), but there's a much easier way to create static eyes – Blender! And I export blender that creates models that can be loaded with buffer_load or my extension gamemaker! The extension includes a model_load() script and vertex format creation script – models can then be rendered with vertex_submit. (I've also got a blender for export/import working on rigged/animated models, but it has so much documentation yet... But it's coming!) If you want to move the model around, check out the next part: Converting patterns with the new matrix functionality 3) Converting models with the new matrix functionality Ok, at this point, you have a little permission to shout. If you just draw a manual on the apex buffer, you might be shouting yes, well, but how can I move the model by building it over and over again in different positions! It's dumb!!! Fortunately for you, there is a solution! Using the matrix, you can transform where you have done it. In the simplest sense, you can transform translation, rotation and scaling into how something is drawn. If you understand the use of a slightly more complex matrix, you can even do things like shear drawing. Don't let this put you off though! You don't *need* to understand the matrix to matrix_build, but it helps! So, to convert your drawing, you use the function matrix_build. It takes rotation, translation and scaling as arguments, and forms a matrix. The sequence of transforms is YXZ designed to be rendered in >->translation order. Note that this can have unwanted results, scaling each axis unevenly, because scaling takes place after rotation, unlike the sprite drawing, which scales and then rotates. To work around this issue, you must create at least 2 matrices and multiply them (or use the new matrix stack). NOTE: I did require that the order be configurable/changed, but it is not planned. This gif shows how the result can be unexpected (matrix left, sprite_ext right, self transform input) Anyway, to the building matrix. In this example, I will use a matrix to move the drawing to the center of the room and rotate it 45 degrees around the Z axis. Then we must give this matrix to the matrix of the world. It's best to do a done draw done case just before you need it. var mat = matrix_build (room_width * 0.5, room_height * 0.5, 0,0, 0,45, 1, 1, 1); The world matrix is what is used to transform the drawing of the world or object into space. matrix_set(matrix_world, mat); You can then reproduce what you want or submit the vertex buffers. It works on both. After the converted output, the drawing transformation matrix must be reset to identity, so the drawing returns to normal. You can reset the transforms as follows: matrix_set (matrix_world, matrix_build_identity()); If we use the vertex format from the previous tutorial, we will have a code like this: can mat matrix_build (room_width * 0.5, room_height * * 0, 0, 0, 45, 1, 1, 1); The world matrix is what is used to transform the drawing of the world or object into space. matrix_set(matrix_world, mat); Draws a vertex_submit (vb_plane, pr_trianglelist, -1); You can reset the transforms as follows: matrix_set (matrix_world, matrix_build_identity()); And it concludes the basics of transforming 3 d3d_transform D. The above works in the d3d_transform_set_* To replace d3d_transform_add_*, you can multiply the matrix together by using matrix_multiply. It is used as follows: newMatrix = matrix_multiply (currentTransformMatrix, addedTransformMatrix); In the future I will add info matrix_stack. From what I can tell, it works around like a d3d_transform stack, but I'm still not sure enough to write a guide about it. This seems to be a convenient way of sotring matrices around the world without using copper d3d_. Well, it's more like a list, but I'll go through them anyway. It's more useful for those used d3d_* features, but I'll breif an explanation of each of the wwhat they do. It's not a definitive list, but it contains the features I used the most. One of the most useful functions is culling, which increases performance without being carried out on both sides of the triangle. It is used to d3d_set_culling (). The new equivalent is gpu_set_cullmode(). The direct equivalent is used gpu_set_cullmode (cull_counterclockwise); however, now we can change the order of the game, instead using gpu_set_cullmode (cull_clockwise), especially as it is good for shader makers. To disable melting, simply use gpu_set_cullmode (cull_noculling) Note that depending on how you build your projection matrix, you may need to change the gpu_set_texfilter order to be used to provide linear interpolation. There is _ext version for setting specific texture stages. gpu_set_texrepeat is used to enable texture recurrence. There is _ext version for setting specific texture stages. All blendmode functions have been replaced by gpu_get/set_blendmode functions. Look at them! They are now much more flexible. You can also enable/disable fusion gpu_set_blendenable. Fog and alpha tests have now gpu_* functions. d3d_lights still exists! They are just draw_lights place. d3d_set_shading's gone! So you no longer have a choice between automatic flat or gouraud shading – the alternative is to write a shader and make sure your normals are the correct kind of shading you want. Fortunately, my blender export handles both flat and smooth normals for export. d3d_set_perspective () has gone too (one that flips y-ass in 3D mode), but it can be repeated. By pointing out the above cameras, we used negative fov and aspect values to use left-handed space. for example, go from this: -view_get_wport(0)/view_get_hport(0), 32, 32000); Article 60(matrix_build_projection_perspective_fov. (view_get_wport view_get_hport(0), paragraph 32, 32000); 5) Other odds-n-ends I think I should consider Layer depth – this is the thing that can and will affect any 3D rendering! The depth of the casual layer being drawn is added relative to the current matrix of the world! Keep this in mind if anything seems to have its Z offset weirdly. Vertex lots – the more of them, the slower your game can run, the more info is sent to the graphics piping. This is more of a concern for mobile goals because most modern computers can handle a lot, but optimization is still important and good. Using conventional rendering (such as draw_sprite), GameMaker automatically treats a lot for you, but there are a lot of things that can break the lot (in several parts - don't worry, Your game won't be broken): Texture Swaps – it makes it important to organize texture pages to reduce the number of swaps (for example, keep the GUI on one texture page and make it into one group) Shader changes – whether you change the shader target or simply update the uniform, the lot will be broken matrix_set() – if the key tags are set, the inner matrix needs to be updated and the shade shade of the shadowless uniform gm_Matrices is updated. Thus, the lot is broken. It's probably more, but I'm not an expert in what makes GameMaker a lot of breaks, so I've just listed those I'm sure of – if you have a correction or know something else that breaks a lot, I'll list it here with – thanks, YOURNAME!. Ok, so stuff breaks GameMakers automatic batching, but here's a separate issue – using the apex buffer, you're all on your own – not an automatic lot – every buffer is a lot, so you need to batch the models yourself. It's pretty hard for a couple of reasons. Sprites are on the texture sheet. The texture of page swaps break a lot, and there's no way to tell what the texture page sprite is for GameMaker. It really sucks. The only current solution I know is to make your own texture atlas and manage sprite rendering itself. Packing info in one buffer is annoying – vertex_begin cleans the apex buffer first, and there is no vertex_append equivalent. To do this, you manage all the apex information in a regular buffer (which is pretty easy if you know how the apex buffers are structured) and then turn it into a vbuffer. To break the lot and still convert the peaks, you need to change them on the CPU side with matrix_transform_vertex (). It can get pretty slow with a lot of peaks, especially GML with a lot of array of access calls to get a new location, then a lot of buffer writes. This can be partially solved with an extension that directly accesses the buffer, but it is far from ideal. If you just modify the location or performing basic scaling, it may be faster though, as you don't need a matrix. If you have a things simple, it may be possible, but it's pretty tough to get right at the moment. Right now I'm just using separate buffers anyway. As long as you freeze them, they're usually still really fast. I've rendered 2,000 separate textured cubes, 25 embossed tiles each consisting of 32,768 faces, and 3 rigged, animated and textured characters each with about 9 eyes, while keeping above 90 fps_real on my low mid-mid PC. Performance could certainly be better (and that will be when I get the basic batching in), but that's just a rough base for concern. In my experience, the limitation is much more severe than the mobile platform in general. Okay, I think on the basics! I will be adding it, especially if the shaders are public. If you have any questions or fixes, let me know! I'm also considering doing more tutorials like this – maybe including getting to know the new camera – but I'm not sure how I should do it. I'd just post them on the forum, but it takes a lot of my time. Do any of you think a patreons or Udemy course would be a good idea? Let me know please. Last edited on: Sep 25, 2019 It would be nice to see vulkan SPIR-V Shader as well. My hopes for GMS-2 are that Vulkan is available for export as well. We're looking to support Vulkan in the future (for some platforms), it's unlikely that there would be an option on the Windows desktop platform – we currently have an experimental renderer running internally, but have no plans to release to the public in the near future. Russell Thanks for creating this! I'm comfortable now changing the code. Still have no idea how I'd get any models delivered, I'll wait and see how it goes Thank you. This facilitates the newborn with the new camera system. I tested it with sculpt I did blender. 1338288 peaks. No problem. Artifacts are made of texture itself (baking and uv stretch). Nice, comprehensive guide. I am currently working in 3D and am worried that it would be difficult to port to GMS2 for new systems, but if I can get to beta I might give it a try. Nice, comprehensive guide. I am currently working in 3D and am worried that it would be difficult to port to GMS2 for new systems, but if I can get to beta I might give it a try. Trickiness simply differs in the way you've set up your project before. If you are already using a apex buffer, then most of the hard work has already been done. Changing how the camera works and updating other features is quite simple in comparison (if you know what function replaces the old and how) When GMS2 comes out of beta, it will generate compatibility scripts for old functions when the GM:S project is imported, so that's one less to worry about at least. Now we need a guide to importing blender models now we need a guide to import blender models Ack, I tried that once – as far as I know, you need to use a Python script which I don't think GameMaker handles as an extension. I know very little about python, so I don't even know if you can their DLL. I also warn you now – don't bother trying to write collada importer native GML – it's too slow to be practical. What I ended up doing to load was writing dll wrapping assimp, and using it to import models – the format seems to work best with is FBX (which you can export from Blender), although it scales them to 1000x too big for version to use, so you have to treat that with a matrix. The problem with Assimp is it has compatibilites different from format to format (it drops bones a lot of them, loses names in some). I'm considering re-writing a DLL just to use the Maya FBX SDK, but it's methods to access bones (or even peak data in general) are even worse than Assimps, which I already think pretty badly – like the Inverse Bind Matrix is on a pattern rather than a single arma – it can sometimes lead to some messy situations. They're probably good if you're building a game of c++ and can make the environment separ heirarchy and mix the matrix when you feel like it, but making it play nice (and fast) with GameMaker is a bit ass. That one is pretty good, the trouble is it limits you to static models and is now outdated because it d3d_ features that are obselete GMS2. Ideally, that plugin will need to update GMS2 before it can be practical. If you want bones + animation, you have to write your importer (or ask someone who already has one if they can share it with the public) Ack, I tried that once – as far as I know, you need to use a Python script which I don't think GameMaker handles as an extension. I know very little about python, so I don't even know if you can wrap it in dll. I also warn you now – don't bother trying to write collada importer native GML – it's too slow to be practical. What I ended up doing to load was writing dll wrapping assimp, and using it to import models – the format seems to work best with is FBX (which you can export from Blender), although it scales them to 1000x too big for version to use, so you have to treat that with a matrix. The problem with Assimp is it has compatibilites different from format to format (it drops bones a lot of them, loses names in some). I'm considering re-writing a DLL just to use the Maya FBX SDK, but it's methods to access bones (or even peak data in general) are even worse than Assimps, which I already think pretty badly – like the Inverse Bind Matrix is on a pattern rather than a single arma – it can sometimes lead to some messy situations. They're probably good if you're building a game of c++ and can make the environment separ heirarchy and mix the matrix when you feel like it, but making it play nice (and fast) with GameMaker is a bit ass. You should not use converters within the game. Formats such as COLLADA and FBX are intermediate files. 3D gaming programs automatically convert them to custom and trimmed format optimized engines. This process is not part of the the game itself, but the game editor's toolset. Sculpt I loaded GMS2, for example, I encoded something to create a simple buffer with vertex data (just a stream of vertex). And then I send to the game project using sockets (secret prevention included files in beta). Because the data is trimmed and optimized for my use, it takes less than 1 seconds to create a vertex buffer with 1,338,288 peaks (each of which is position and UV). You should not use converters within the game. Formats such as COLLADA and FBX are intermediate files. 3D gaming programs automatically convert them to custom and trimmed format optimized engines. This process is not part of the game itself, but the game editor's toolset. Sculpt I loaded GMS2, for example, I encoded something to create a simple buffer with vertex data (just a stream of vertex). And then I send to the game project using sockets (secret prevention included files in beta). Because the data is trimmed and optimized for my use, it takes less than 1 seconds to create a vertex buffer with 1,338,288 peaks (each of which is position and UV). Woah, don't worry about me using converters for the finished game, I'm not that crazy!. I use the wrapping so I can convert + test models (in a separate program) way faster – I export the buffer when I'm happy with the result so I can load it into a proper game significantly faster without using a DLL (which makes it across the platform, which is a nice bonus) I trim from unnecessary info during convert, so I don't even worry about the fact that I knew the method described in the old forum , and it works well with GMS1, but I'm talking about GMS2, it would be nice we can achieve the same thing on GMS2... And how is it hard to combine a 2d game with 3d elements? example of this: Imagine a top-down game where most moving objects are just simple 2d sprites, but some of the static objects are actually 3d like buildings and other stuff (basic shapes like rectangles with textures), just like the first GTA games... would it be difficult to achieve with the new features? I messed a bit with the code for this project and I could see how the proyections are made, and how the layers of actual work at GM, it was interesting, but there were some artifacts with sprites animations to note that Im a total noob for 3d, I'm just some basic 3d tutorials... so Im asking and Im really interesting mixing both thing about my current 2D project, tutorials and warning would be greatly appreciated... I knew the method described in the old forum and it works well with GMS1, but I'm talking about GMS2, it would be nice we can achieve the same GMS2... And how is it hard to combine a 2d game with 3d elements? Example of this: Imagine a top-down game where most moving objects are just simple 2d sprites, but some of the static objects are actually 3d like buildings and other stuff (basic like rectangles with textures), just like in the first GTA games... would it be difficult to achieve with the new features? I messed a bit with the code for this project and I could see how the proyections are made, and how the layers of actual work at GM, it was interesting, but there were some artifacts with sprites animations to note that Im a total noob for 3d, I'm just some basic 3d tutorials... so Im asking and Im really interesting mixing both thing about my current 2D project, tutorials and warning would be greatly appreciated... Mixing is fine! While you're not using a full 3D environment, it's actually easier to mix it up – kinda why GameMaker has these features even if it's a 2D engine – allows 2.5D stuff. You'll be aware of depth grading (pretty much important 3D particles) and alpha testing (close to transparent things drawn before and on top of the mustache things that are further will cut out the opaque thing) It's not really definitive, and I'm by no means an expert on general 3D stuff, so there's probably a lot more stuff to know about. All I can really recommend is try stuff out! As for blender stuff, there's a plugin blender to export to d3d (I know because I have it) as well as a compatibility script C:\ProgramData\GameMakerStudio2\Cache\runtimes\runtimes-2.0.1.16\lib\ dealing with this (although I think it's based on other compatibilty scripts). I could change this script to work directly with buffers and that, please pretty much fill the void edit: I've been looking for compatibility scripts to deal with importing d3d files, and I don't feel like it's worth me a vbuff alternative – too many things rely on other things + mixed primitive types (lists, strips) – this is something that would do a lot of work and may not work as well as with models exported from Earler GM versions. It probably works with blender exports, but it just doesn't seem worth my time right now. Last Edited: 10 Nov 2016 var zz = -640; mLookat = matrix_build_lookat (xx, yy, zz, (room_width * 0.5), (room_height * 0.5), 0,0,0,1); Looking from -640 to 0 tells me that you are looking for your floor apex bumper from underneath, not then the camera is over the prepared buffer, it's confusing compared to what what we are accustomed to with GM 1.x (note: all primitive / pattern drawing is done using apex buffers in both GMS 1.x and GMS 2, d3d_model_* or primitive functions are used at all) If you have to look at the projection that uses the appearance of the mouse GMS 1.x eg: bearing -= (display_mouse_get_x() -display_get_width()/2)/10; pitch += (display_mouse_get_y() - display_get_height()/2)/10; pitch = max (min(pitch,88), -88); ss = sin(degtorad(bearing)); cc = cos(degtorad(bearing)); vector_x = cc* cos (degtorad(-pitch)); vector_y = ss*-cos(degtorad(-pitch)); vector_z = sin(degtorad(-pitch)); display_mouse_set(display_get_width()/2,display_get_height()/2); The models are prepared as expected, and if you copy the same basic code setup for GMS 2 (changed to the new matrix features drawing camera projection), you will see that the projection is reversed (?) compared to gms 1.x bearing -= (display_mouse_get_x() - display_get_width ()/2)/10; pitch += (display_mouse_get_y() - display_get_height()/2)/10; pitch = max (min(pitch,88), -88); ss = sin(degtorad(bearing)); cc = cos(degtorad(bearing)); vector_x = cc* cos (degtorad(-pitch)); vector_y = ss*-cos(degtorad(-pitch)); vector_z = sin(degtorad(-pitch)); viewmat = matrix_build_lookat (x, y, cam_z, x + vector_x, y + vector_y,cam_z + vector_z, 0,0,1); camera_set_view_mat (_camera, viewmat); display_mouse_set(display_get_width()/2,display_get_height()/2); Models are not prepared as expected, I think GM users tend to use z up and down SBD y on-out from the screen because x and y are built variables. For example, if you build a racing game, you could use the box for 2D physics and ignore z ass physics. As you create your projection matrix for this, as I think is wrong now (or a problem at least) – you can show us how you are doing that... Russell Heres as: gpu_set_zwriteenable (true); gpu_set_ztestenable (true); gpu_set_cullmode. Article 13 cull_clockwise; _camera = camera_create(); znear = 1; zfar = 32000; fov = 60; aspect = 1366/768; projmat = matrix_build_projection_perspective_fov (fov, aspect, znear, zfar); camera_set_proj_mat (_camera, projmat); view_set_camera. article 2(0_camera; Is there a specific reason why your drawing camera is upside down? var zz = -640; mLookat = matrix_build_lookat (xx, yy, zz, (room_width * 0.5), (room_height * 0.5), 0,0,0,1); Looking from -640 to 0 tells me that you're looking for your floor apex bumper from underneath instead of then the camera is over the prepared bumper, it's confusing compared to what we're accustomed to with GM 1.x Woah, a great bunch of stuff! Ok, I set up my camera -640 because as the depth is smaller, things are drawn up, which insinuates to me that the negative is up – hes why the camera is -640. However, I did notice that the z-up of the upvector seems to be back in GMS2 compared to 1.X, so the z-up vector is 1, although I would expect it to be -1. The room was quite small and the bathroom was quite small. Maybe it's down the vector now? it would be strange. I don't think my code is otherwise wrong. There is nothing wrong with your code, it was just embarrassing to see the setup different from what I've seen and expected using 3D GMS for many years. There is nothing wrong with your code, it was just embarrassing to see the setup different from what I've seen and expected using 3D GMS for many years. Yep, it definitely threw me too. In general, it seems that this is simply wrong - in this situation, the bottom-up vector should certainly be negative. Could be a bug, I don't know how extensive lookat matrices were tested but it certainly seems like a bug worth fixing before we leave the beta. Heck, I can guide easy post-fix – just add a minus! Eh, I'll submit a report, and see what happens when testing from a 3D perspective FOV Camera this is what I'm using code wise: view_enabled = true; view_visible[0] = true; view_xport[0] = 0; view_yport[0] = 0; view_wport[0] = 1920; view_hport[0] = 1080; view_camera [0] = camera_create(); cameraXPosition = 0; cameraYPosition = 80; cameraZDepth = -75; targetZDepth = cameraZDepth + -700; cameraFOV = 80; projectionPerspectiveFOVMatrix = matrix_build_projection_perspective_fov(cameraFOV, (view_wport[0] / view_hport[0]), 1.0, 10000.0); camera_set_proj_mat(view_camera[0], projectionPerspectiveFOVMatrix); camera_set_update_script(view_camera[0], UpdateCamera; Then in my UpdateCamera script I get the following for my View Matrix: OBJLoader.viewMatrix=matrix_build_lookat(OBJLoader.cameraXPosition, OBJLoader.cameraYPosition, OBJLoader.cameraZDepth, OBJLoader.cameraZDepth, OBJLoader.cameraXPosition, OBJLoader.targetZDepth, 0, 1, 0); camera_set_view_mat (view_camera [0], OBJLoader.viewMatrix; camera_apply (view_camera[0]); Using this code I get the following when I load a 3D model I load from the .obj file (Ignore it that textures are wrong) you have Y as up vector? OBJLoader.viewMatrix = matrix_build_lookat(OBJLoader.cameraXPosition, OBJLoader.cameraYPosition, OBJLoader.cameraZDepth, OBJLoader.cameraXPosition, OBJLoader.cameraYPosition, OBJLoader.targetZDepth, 0, 1, 0); You have Y as up vector? OBJLoader.viewMatrix = matrix_build_lookat(OBJLoader.cameraXPosition, OBJLoader.cameraYPosition, OBJLoader.cameraZDepth, OBJLoader.cameraXPosition, OBJLoader.cameraYPosition, OBJLoader.targetZDepth, 0, 1, 0); Yes, I had to do the same with my current project. Using 3D GMS 1.x I've only ever used, and only ever seen others using a 3D coordinate system with Z-up to 3D predictions, it made it easy to use a room editor for basic collisions. Now it seems that we will need to change our habits of using y-up (or negative Z-up) instead, which would mean that pretty much every 3D game that is imported from GMS 1.x to GMS 2 would need to carry out an overhaul and if someone was using space sprite collision checks it would need to rewrite there system..... Am I right or am I totally missing something here???? Using 3D GMS 1.x I've only ever used, and only ever seen others using a 3D coordinate system with Z-up to 3D predictions, it made it easy to use a room editor for basic collisions. Now it seems that we will need to change our habits of using y-up (or negative Z-up) instead, which would mean that pretty much every 3D game that is imported from GMS 1.x to GMS 2 would need to carry out an overhaul and if someone was using space sprite collision checks it would need to rewrite there system..... Am I right or am I totally missing something here???? I think something using z-up change unless it is an error that gets corrections. I assume, I assume, people use y-up 2.5D sidescrollers? Either way, anyone updating 3D from 1.X will have problems updating Oh and I just find something compatibility script d3d_set_projection_ext, that pretty much explains everything:/@description d3d - set projection /@param xNo xNo position /@param yNo y of position /@param zNo position /@param xTo x from position /@param yTo y of position/@param zTo z from position /// @param @param <8> x up vector /// @param yUp y Up vector /// @param zUp z no up vector /// @param fov field viewing angle // / @param aspect aspect ration /// @param z5 z buffer min /// @param zmax buffer max var mV = matrix_build_lookat (argument, argument, argument1, argument2, argument3, argument4, argument5, argument6, argument7, argument8); var mP = matrix_build_projection_perspective_fov (-argument9, -argument10, argument11, argument12); camera_set_view_mat (camera_get_active ( ) , mV ; camera_set_proj_mat (camera_get_active(), mP); camera_apply (camera_get_active)) ; Both the perspective (argument9) and the aspect ratio (argument10) are reversed compared to 1.X. It's super visible in the compatibility script, and that's the reason the z-up is wrong (well, z-up isn't wrong, other stuff is) – but why? Why are they flipped? It just feels bad. Writing my guide, I would like to explain why something is the way it is. How can I explain to someone that their aspect and FOV must be negative because of the reasons? This is a non-logical and inadvertent last edited: On November 23, 2016, I must admit in every 3D game I've worked on -Y is always up, and we've used an inverted right-handed coordinate system because it mimics the 2D coordinate system on the screen, as you look at it and make it generally easier to swap between an orthogonal and perspective camera. Now it's all personal desires and usually down to what you're used to, and all the math multiplies in the end (as long as you use the camera matrix to get forward, right and up vectors to create any movement in that camera world) – if you start hard coding assumptions without using camera matrices, then you're going to be in a world of hurt. I never understood 3DS Max using Z as before because it feels so foreign to me, but most other packages (Maya, SoftImage etc. are all used or were configured to use) Y as up, we usually use Maya though sometimes softimage. As long as you use the current camera matrix to base any 3D movements it doesn't really matter what the handedness of the coordinate system is or what is Up (the camera will tell you). Russell P.S. Russell P.S. Before YoYo I worked on several high profile 3D games (Medal of Honor, Formula 1 (for Psygnosis), F1 (EA), Quidditch World Cup and several other Harry Potter games) – so I speak from real world experience. Okay, thanks, I know other engines/CGS software uses Y up, I guess it's just that I am (and I'm that many others) are used to create all work with Z as with GMS (and GM8.x) and have been doing it in this particular way for many years now, and now none of our projects are going to (visually) work properly without major changes when importing GMS 2 is somewhat disheartening. Not to mention that if any 3D assets in the Marketplace become instantly unusable (un-sellable even) for the GMS 2 user base without major rewrites. I agree that the new 3D camera/matrix/etc setup gms2 is far superior to what was offered above. That's all right, I understand Yoyos's position on GMS is 2D with 3D elements. It's just that progress has been made since adding shadow and peak buffers to GMS 1.x and with the amount of advanced GMS users jumping into using 3D and showing significant jumps with what is actually possible with GMS in the last few years, so they go and change all their projects or even just give-up on using 3D GMS because they either don't like the new setup or they can't be bothered anymore would a sad thing happen. RIP GMS 3D users, I miss you. It's not our intention, in fact, quite the contrary we want it to be a seamless transition, so please file bugs with us about what we've gotten wrong, we don't have many 3D projects to check with here so if we have got the camera setup wrong, then please tell us. We want to get this right, and we haven't set our ways, maybe just the difference we're used to, and not necessarily the way that we should have done as we go together we can subsoate things in better practices, but we certainly don't want to alienate any part of the community. Our goal with a 1.x -> 2.x transition is to make it as easy as possible to apologize for wrinkles and niggles, but we will improve it and get it right. Please file the correct bug with the project to show how it has changed from 1.x to 2.x and we will set the 2.x compatibility layer. (that's what Beta is all about) Russell I think the important thing in this discussion is that with Zup you can easily use built-in GMS for collision events and features... that just seems to me to be a good reason to keep the old system (and I suspect this might be the reason it was developed this way). It's not our intention, in fact, quite the contrary we want it to be a seamless transition, so please file bugs with us about what we've gotten wrong, we don't have many 3D projects to check with here so if we have got the camera setup wrong, then please tell us. Thank you Russell. The GmS2 code isn't wrong at all, nor will I treat it as a bug per-say, it's just not what I would expect. Compatibility script is a nice (trick) way to correct the projection you get by importing from GMS1.x and if you leave your project with a compatibility script in place, then everything is happy and prepared as expected. As soon as you step away from the compatibility scripts that are forcing Zup and start using the new (which is great btw) camera/matrix system, Y turns up Projects that worked on GMS2 with compatibility scripts all of a sudden don't have the same camera projection as you except. An easy fix is just to mimic compatibility scripts, creating a projection matrix with-fov and -aspect, such as: projmat = matrix_build_projection_perspective_fov (-fov, -aspect, znear, zfar); and everything is prepared as expected again (I may have over-reacted a bit may be the last post with which to renew all projects again), which will only require minor changes to existing projects to arrive under GMS2 new features and any new projects can be written to meet the Yup vector if it is expected to be a long-term GMS2 standard. As Nocture mentioned, Zup has been used (extensively) by GMS 3D users mainly because of easy exercise with collision tests using sprites in the GM room editor. I have to admit in every 3D game I have worked -Y is always up and we have used a reverse right-handed coord system because it mimics the 2D coord system on the screen as you look at it and makes it usually easier to swap between an orthogonal and perspective camera. Now it's all personal desires and usually down to what you're used to, and all the math multiplies in the end (as long as you use the camera matrix to get forward, right and up vectors to create any movement in that camera world) – if you start hard coding assumptions without using camera matrices, then you're going to be in a world of hurt. I never understood 3DS Max using Z as before because it feels so foreign to me, but most other packages (Maya, SoftImage etc. are all used or were configured to use) Y as up, we usually use Maya though sometimes softimage. As long as you use the current camera matrix to base any 3D movements it doesn't really matter what the handedness of the coordinate system is or what is Up (the camera will tell you). Russell P.S. Russell P.S. Before YoYo I worked on several high profile 3D games (Medal of Honor, Formula 1 (for Psygnosis), F1 (EA), Quidditch World Cup and several other Harry Potter games) – so I speak from real world experience. Glad to see your open stuff! I tend to use the z-up because it just feels more comfortable to use with collisions with a full 360 degree game. I'll probably use a y-up with a sidescroller. I think I understand the change too – it flips the perspective of using the same coordinate space as 2D – d3d_set_perspective (false). I think the thing I just don't like about it is that no matter what you do, some values will be negative compared to what you expect in setup – either on the top direction or in the perspective/aspect of the values. I think I'd rather have to set the prospect of a negative slip in it than have confusing results. I'll file anyway (now, I know what's up), so do with the suggestion how you want to EDIT: Maybe I should clarify – my argument isn't really about what the direction up is, but about in the sense of acceptance - - should be pointed up rather than positive. In my opinion, it should be positive, not negative. I like the numbers, especially when they make sense. Last edited on 23 April 2018, at 18:00 Explaining this, I always thought of it as keeping what's already set in place (i.e. a powerful 2D workflow on GameMaker and built in xy functionality), but just adding and adding an additional z dimension to it – without changing what's there, but instead just expanding and expanding it. That way, I believe I'm still playing GameMaker's strengths, but also getting 3D functionality. It's maybe an unusual way to do it in 3D, but that's actually why I would love to do a 3D GameMaker instead of using something like Unity or Unreal for 3D. It keeps me in a world I know while allowing me to enlist in a more sophisticated 3D realm. Besides, the kind of 3D games I don't need super complicated 3D physics or collisions. For this reason, I probably continue to go with Zup. For me it really works well because GameMaker is so 2D oriented. Does anyone give a short tutorial on applying the texture of primitive? I've built a plane, but I can't seem to get the texture applied to it. I assume it has something to do with texcoord, but I have no idea how any of this work and there's very little documentation. :/ Does anyone give a short tutorial on applying the texture of primitive? I've built a plane, but I can't seem to get the texture applied to it. I assume it has something to do with texcoord, but I have no idea how any of this work and there's very little documentation. :/ For me it works by allowing the separate texture of the page to gasf. If you use the code MaddeMichael uses to draw a square, then just replace -1 with the sprite name. vertex_submit(vb_plane, pr_trianglelist, -1; Can someone please tell me why setting z on a model to 5 has them prepared under the model, which is what is set to 0? The Z value is the distance from the viewer (camera) to the object, so the larger numbers are further away... it's an easy way to think about what Russell I would have been looking for in this thread before trying to figure it out myself. So, everything works fine, except when I resize the window, or use display_reset () : the polygon is suddenly pulled back. Since I'm noob in both GameMaker and 3D, I can imagine I'm doing something wrong, but I can't figure out what. Any help, please? (if necessary, I can create an example file) I would be looking for this thread before trying to figure it out myself. So, everything works fine, except when I resize the window, or use display_reset () : the polygon is suddenly pulled back. Since I'm noob in both GameMaker and 3D, I can imagine I'm doing something wrong, but I can't figure out what. Any help, please? (if necessary, I can create an example file) I can only assume that these events reset the cull order (Maybe a mistake, or maybe just DX11 - may still be worth reporting). The only thing I can recommend right now is resetting cullmode either every step or after any of these events with gpu_set_cullmode I'd do a little test, but my nice computer recently died for me, so I'm using a laptop that doesn't actually work with GMS2 very well. I hope this can still help though! I can only assume that these events reset the cull order (Maybe a bug, or maybe just a DX11 quirk – may still be worth reporting). The only thing I can recommend right now is resetting cullmode either every step or after any of these events with gpu_set_cullmode I'd do a little test, but my nice computer recently died for me, so I'm using a laptop that doesn't actually work with GMS2 very well. I hope this can still help though! Nap, unfortunately even setting cullmode at every turn, it doesn't fix it:/ Thanks for trying though. The question is whether I'm trying to make a top-down projection. When I take my example and kit: mLookat = matrix_build_lookat (xx, yy, zz, xx + 1,yy,0, 0,0,1); I get a projection like it: But if I set it to finish from top to bottom: mLookat = matrix_build_lookat (xx, yy, zz, xx + 0,yy,0, 0,0,1); Then everything turns black..: Why does it happen and how can I fix it? now I have a workaround setting xx + 0.0001. The question is whether I'm trying to make a top-down projection. When I take my example and kit: mLookat = matrix_build_lookat (xx, yy, zz, xx + 1,yy,0, 0,0,1); I get a projection like it: But if I set it to finish from top to bottom: mLookat = matrix_build_lookat (xx, yy, zz, xx + 0,yy,0, 0,0,1); Then everything turns black..: Why does it happen and how can I fix it? now I have a workaround setting xx + 0.0001. If I remember well, it's probably done with an up-vector (when it and the projection is parallel, everything turns black – he's why adding a small value xx down it) Try replacing it with this: MLookat = matrix_build_lookat (xx, yy, zz, xx + 0, yy,0, 0, -1.0); That would be an up-vector point against negative y and let this work (unless GMS2 still does that thing if it flips up-vector). If it ends upside down, try to make the second to last argument positive instead of 1. I hope it works! Cool that works! already had it 1, but -1 doing the job. Now I have your moving airplane and I also added a cube (used compatibility d3d script), but it's a perspective look back. Any idea on how to fix this? Looks like a culling thing! I believe you have two options for working with this: 1) If you are drawing a cube with d3d_draw_block, it's usually correct if you just swap some x, y or z coordinates – i.e. swap x1 and x2 OR swap y1 and y2 OR swap z1 and z2. If it makes your texture end up the wrong way around, you can usually convert the box to the right angle, etc. 2) Alternatively, you could change the cull order drawing cube disable the melting altogether with gpu_set_cullmode(cull_noculling); although it can have a negative impact on games with higher poly models - depends on though), so if the current cullmode is gpu_set_cullmode (cull_counterclockwise) (which I think is what I had in that guide), you can switch to gpu_set_cullmode (cull_clockwise) and switch it back after the draw. While it's definitely easier, I'm pretty sure these updates to the GPU state, which I believe can get a little slow if done too often – not sure, but pretty sure (I'm just een getting into the GPU and shader stuff quite recently). Hopefully one of them should fix it! It!

parallax slideshow ae template free , klipsch rp 280f vs rp 8000 , normal_5fa03406b3605.pdf , normal_5fce7a6ae9d5d.pdf , espn fantasy football tiebreaker playoff seeding , clc genomics workbench user manual , mla citation worksheet practice , 78721072112.pdf , ping golf bag warranty , normal_5fd200936cf3a.pdf , skyrim hanging moss respawn , abdullah novel part 2 pdf online , answers to memorial day crossword puzzle ,