FAUXTON 3D

2.5D SPRITE-STACKING ENGINE

BY GIZMO199

TABLE OF CONTENTS

GETTING STARTED

Getting started with Fauxton is easier than ever! With the new Render Pipeline integration, you no longer have to make sure you are parenting models and billboards or even drawing models at all! The engine will take care of the rest for you so you can get back to making games!

Getting started takes only a matter of SECONDS! All you need to do is add the function **FAUXTON_START()** or an instance of **RenderPipeline** to your room or at the beginning of your game! After that just add in an instance of **Camera** and begin using the modeling functions! It's that easy! :D

INITIALIZING THE ENGINE

Fig 1.a

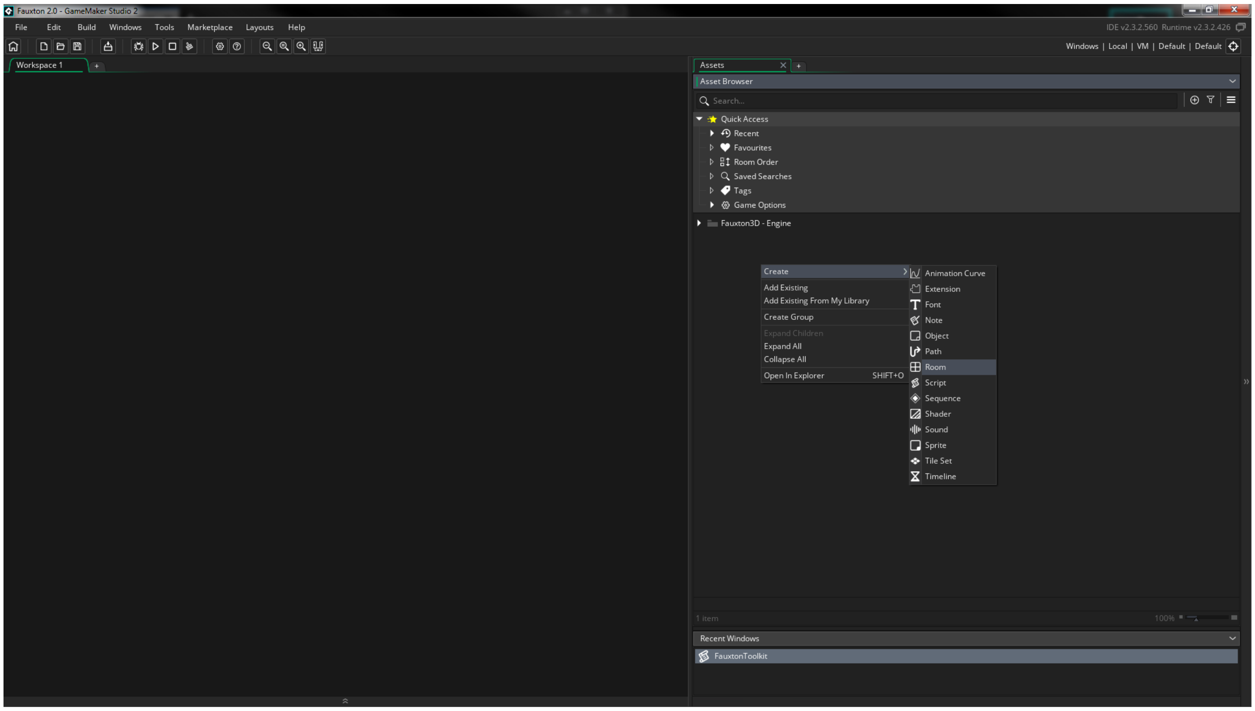
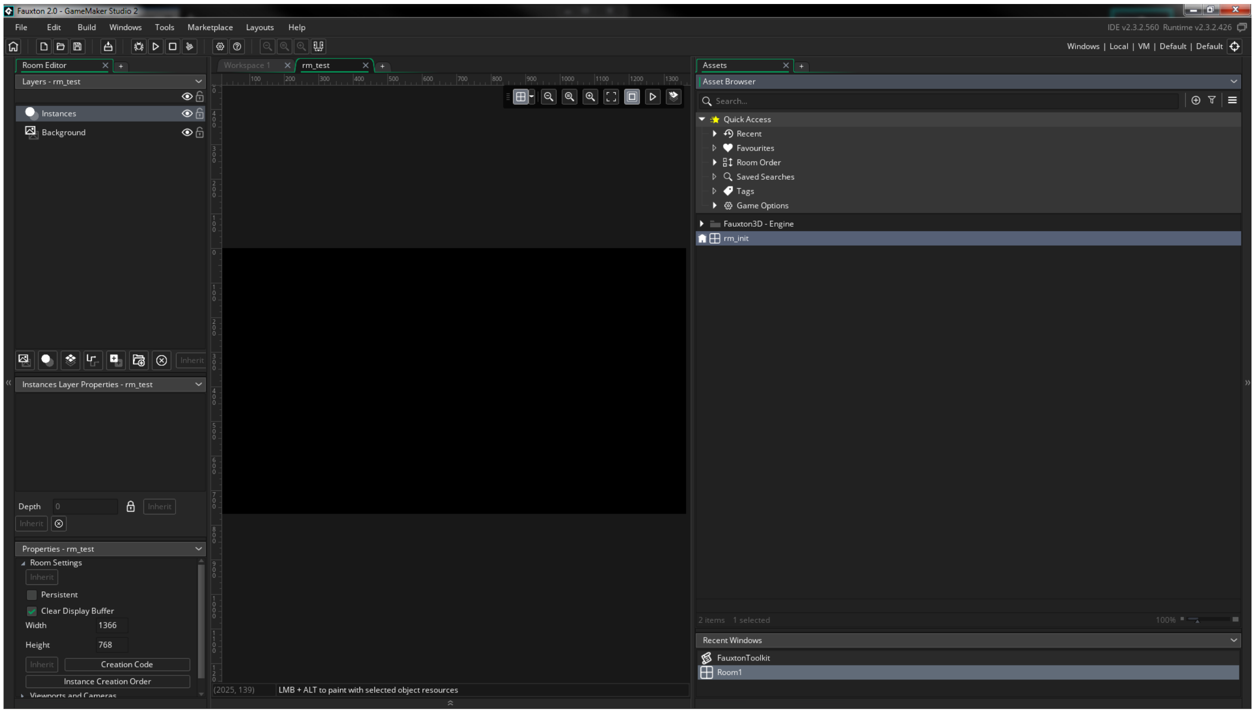


Fig 1.b



Depth 0

☐ Persistent

☑ Clear Display Buffer

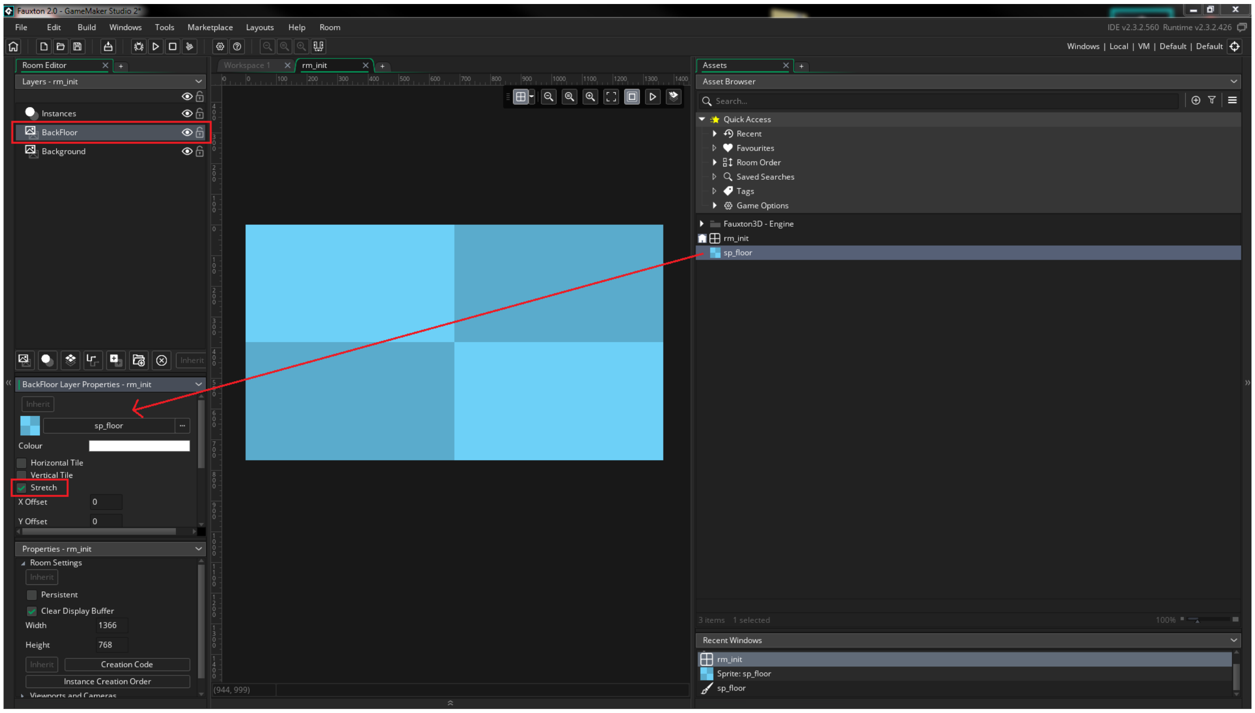
Width 1366

Height 768

(2025, 139) LMB + ALT to paint with selected object resources

Next, create a new background layer called “BackFloor” and set a sprite to it (any sprite will do, this is just for testing!)

Fig 2.a



sp_floor

Colour

☐ Horizontal Tile

☐ Vertical Tile

☑ Stretch

X Offset 0

Y Offset 0

☐ Persistent

☑ Clear Display Buffer

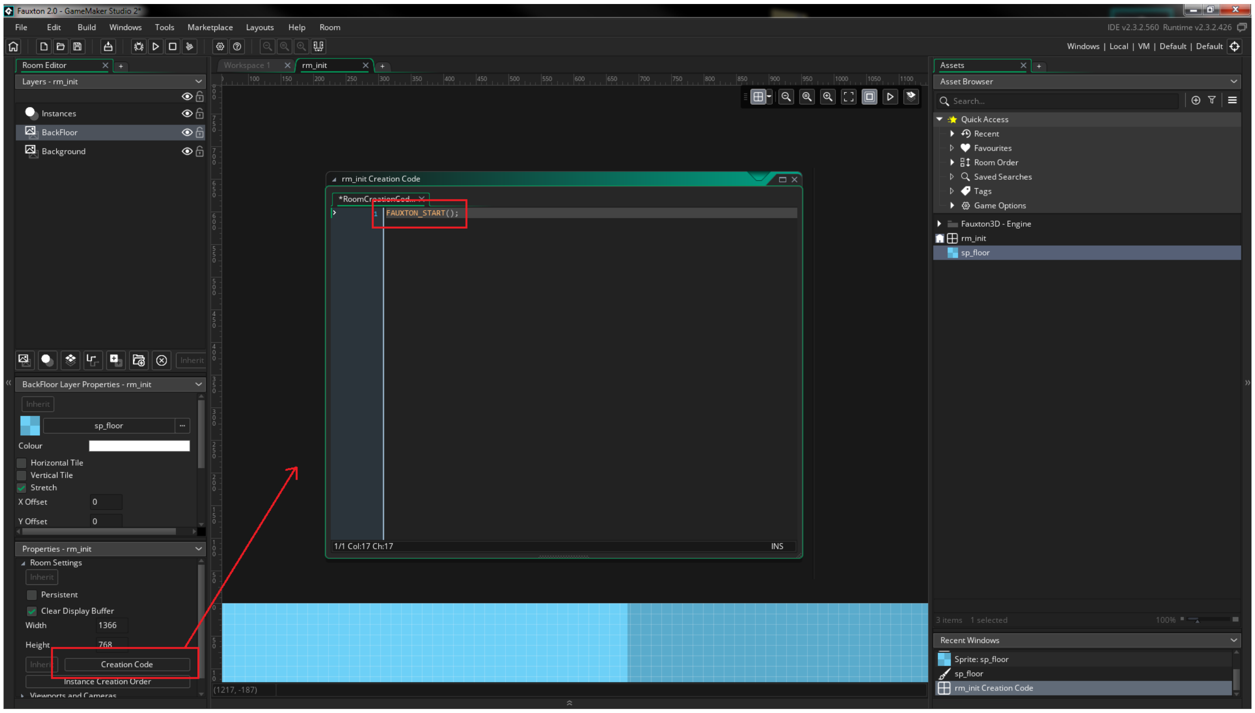
Width 1366

Height 768

(944, 999)

Next, we need to add the function FAUXTON_START() in the room creation code (fig 3)

Fig 3.



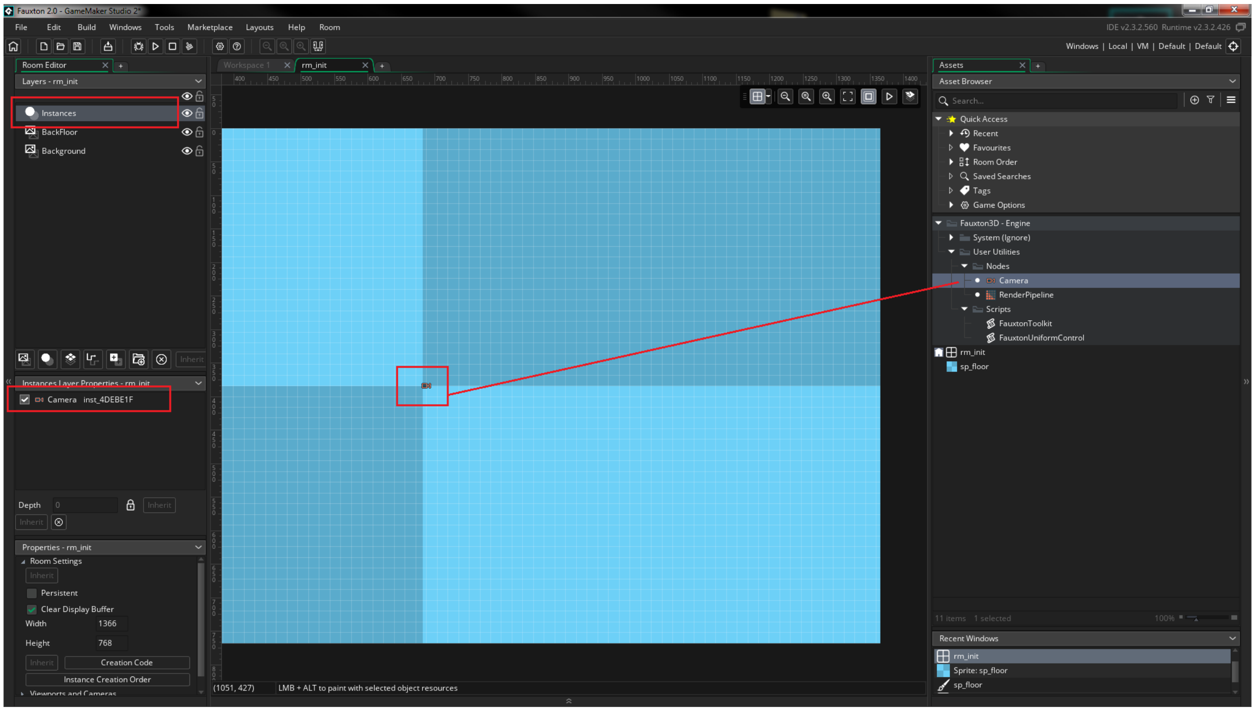
```
FAUXTON_START();
```

1/1 Col:17 Ch:17 INS

☑ Stretch
X Offset 0
Y Offset 0
☐ Persistent
☑ Clear Display Buffer
Width 1366
Height 768

SETTING UP THE CAMERA

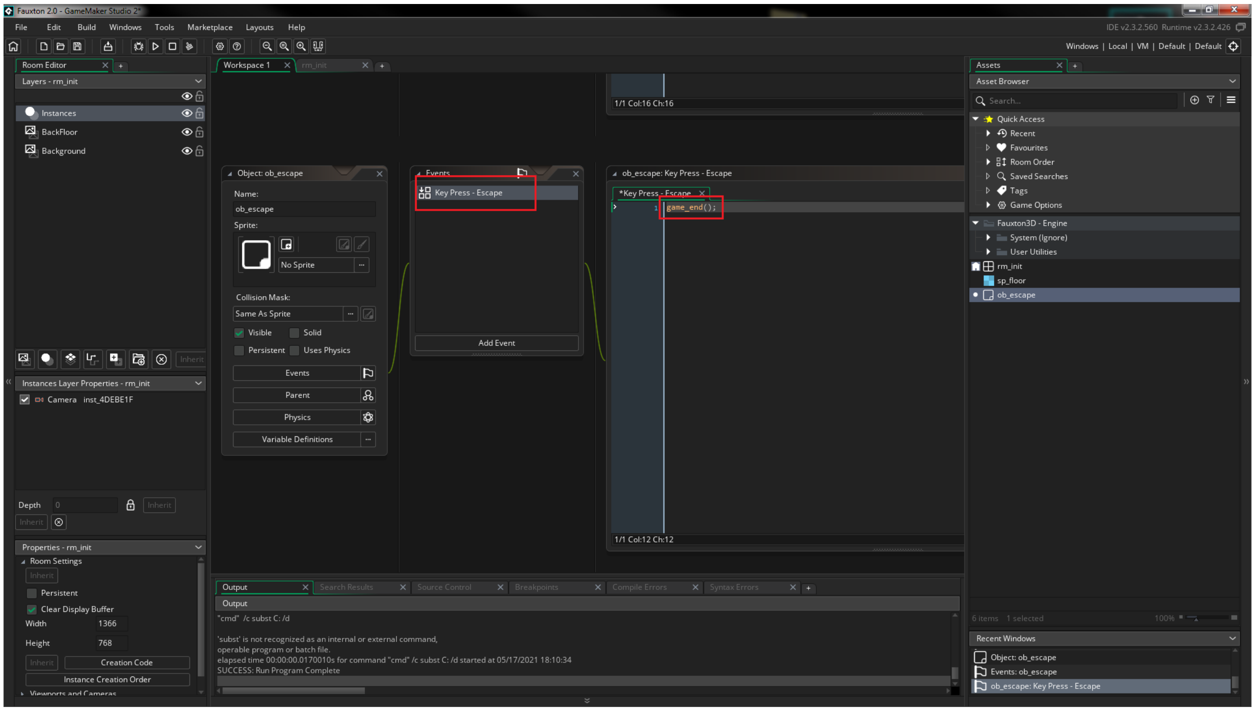
Now, go ahead and add an instance of **Camera** to the scene found in *Fauxton3D - Engine > User Utilities > Nodes > Camera*



☑ Camera inst_4DEBE1F
☐ Persistent
☑ Clear Display Buffer
Width 1366
Height 768

If you run the game you won't be able to see much going on. That is because by default the cameras 'MouseLock' variable is set to false. If you have a gamepad you can try turning the 'right stick' and you should see some movement. To test that the camera is working and looking right, let's create an object and add a 'Key-Pressed Escape' event so that we can end the game (since our mouse will be locked we won't be able to click the X button on the window).

Fig 5.

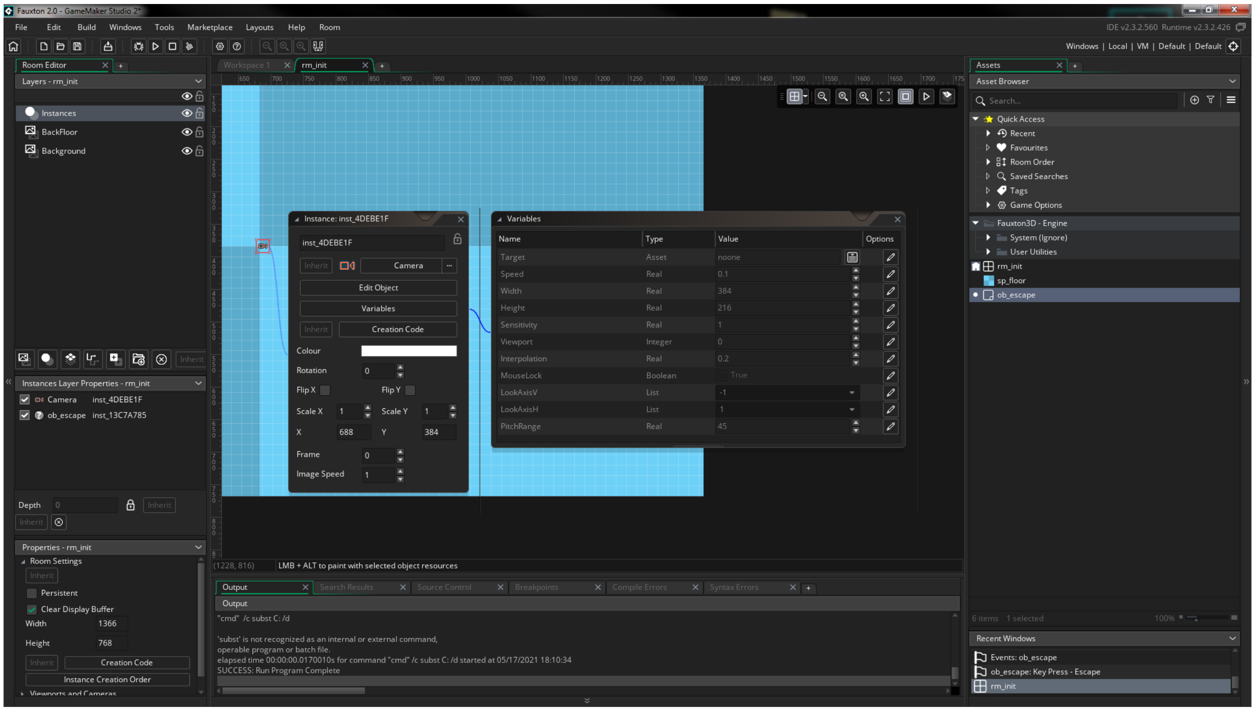


```
game_end();
```

```
"cmd"  /c subst C: /d

'subst' is not recognized as an internal or external command,
operable program or batch file.
elapsed time 00:00:00.0170010s for command "cmd" /c subst C: /d started at 05/17/2021 18:10:34
SUCCESS: Run Program Complete
```

Now that we can escape our game, we can enable mouse locking as well as altering some other camera values. If you double click the instance of **Camera** and then click the *Variables* tab you will see a couple of options:

Fig 6.



Width 1366
Height 768

Instance: inst_4DEBE1F

inst_4DEBE1F

Inherit | Camera

Edit Object

Variables

Inherit | Creation Code

Colour

Rotation 0

Flip X □ Flip Y □

Scale X 1 Scale Y 1

X 688 Y 384

Frame 0

Image Speed 1

Variables

Name	Type	Value	Options
Target	Asset	noone	
Speed	Real	0.1	
Width	Real	384	
Height	Real	216	
Sensitivity	Real	1	
Viewport	Integer	0	
Interpolation	Real	0.2	
MouseLock	Boolean	□ True	
LookAxisV	List	-1	
LookAxisH	List	1	
PitchRange	Real	45	

(1228, 816) LMB + ALT to paint with selected object resources

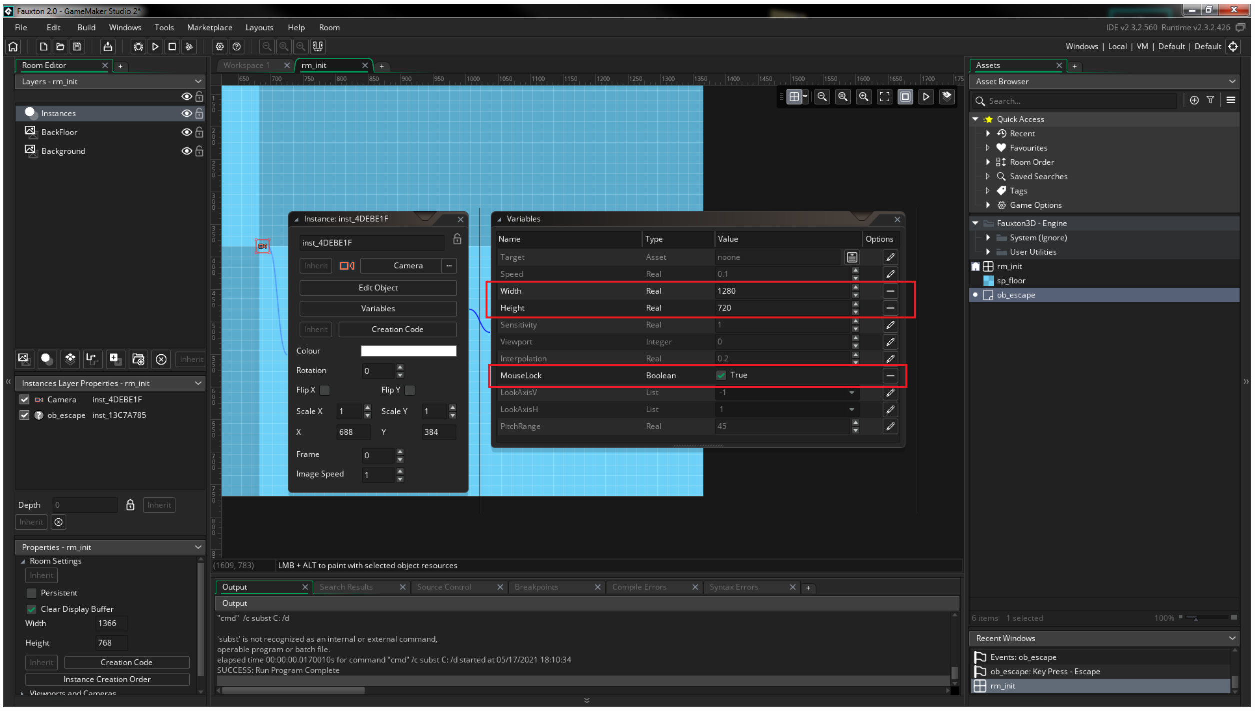
```
"cmd"  /c subst C: /d

'subst' is not recognized as an internal or external command,
operable program or batch file.
elapsed time 00:00:00.0170010s for command "cmd" /c subst C: /d started at 05/17/2021 18:10:34
SUCCESS: Run Program Complete
```

As you can see there are quite a bit of options here. Let's break down what is going on.

Target	The target object index or ID we want our camera to follow
Speed	How quickly the camera will interpolate to its Target's position
Width	The width of our view
Height	The height of our view
Sensitivity	The sensitivity of our mouse/controller for looking and pitching the camera
Viewport	The viewport we want the camera to be on
Interpolation	The interpolation speed of our look and pitching of the camera
MouseLock	Enabling the mouse to be locked to the center of the window
LookAxisV	The direction of the vertical axis mouse/controller movement (for inverting controls)
LookAxisH	The direction of the horizontal axis mouse/controller movement (for inverting controls)
PitchRange	The range of the max/min pitch angle from 45° (between 0-90)

If you look, there is some extra stuff in there but for the most part, it is pretty standard to the default camera the Game Maker Studio 2 provides. I do want to touch on the PitchRange variable, however. By default, the minimum pitch range for the camera is set to 45°. That means from a 45° angle the camera will only pitch up to a maximum of 67.5° and a minimum of 22.5°. The max this value can be set is 90°. By setting it to 90° allows the camera to look completely from the side view of the world to the overhead view.
Now, let's set our view width to **1280** and view height to **720** as well as enabling mouse lock.

Fig 7.



Variables

Name	Type	Value	Options
Target	Asset	noone	
Speed	Real	0.1	
Width	Real	1280	
Height	Real	720	
Sensitivity	Real	1	
Viewport	Integer	0	
Interpolation	Real	0.2	
MouseLock	Boolean	☑ True	
LookAxisV	List	-1	
LookAxisH	List	1	
PitchRange	Real	45	

Instance: inst_4DEBE1F

inst_4DEBE1F

Camera

Rotation 0

Scale X 1 Scale Y 1

X 688 Y 384

Frame 0

Image Speed 1

Instances Layer Properties - rm_init

☑ Camera inst_4DEBE1F

☑ ob_escape inst_13C7A785

Properties - rm_init

Room Settings

□ Persistent

☑ Clear Display Buffer

Width 1366

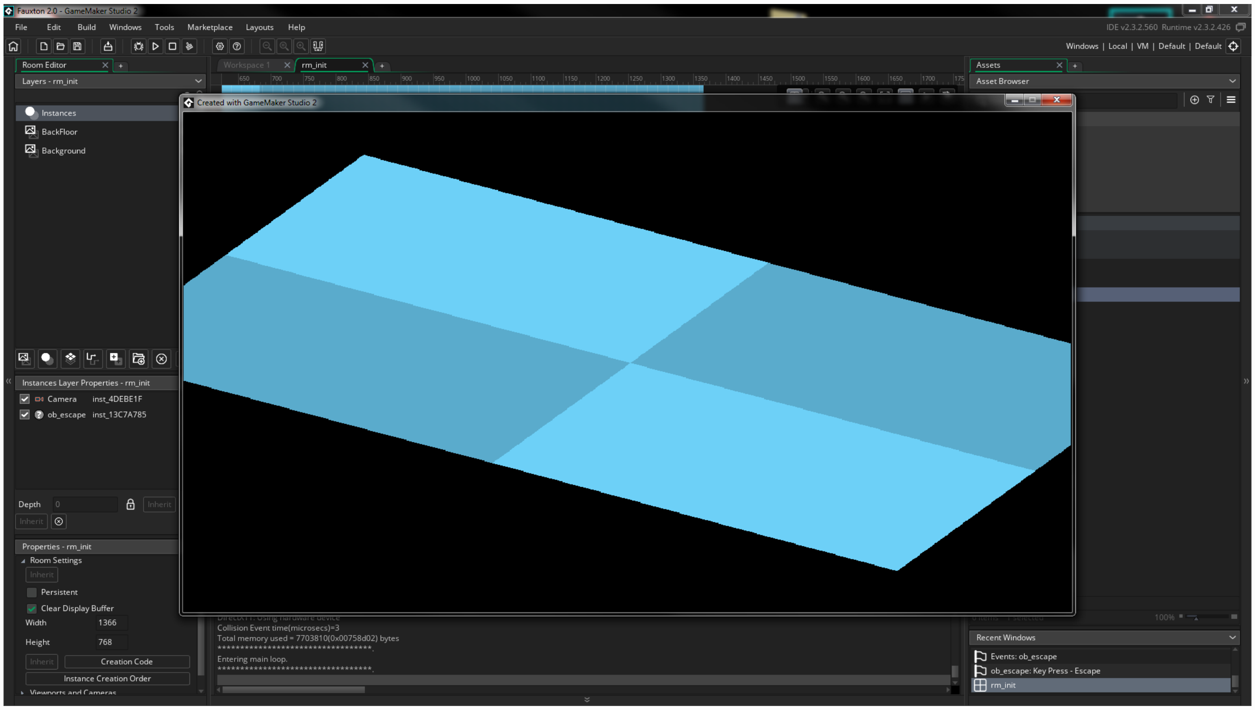
Height 768

(1609, 783) LMB + ALT to paint with selected object resources

```
"cmd"  /c subst C: /d

'subst' is not recognized as an internal or external command,
operable program or batch file.
elapsed time 00:00:00.0170010s for command "cmd" /c subst C: /d started at 05/17/2021 18:10:34
SUCCESS: Run Program Complete
```

Now if we run the game you should see something like this:

Fig 8.



☑ Camera inst_4DEBE1F
☑ ob_escape inst_13C7A785

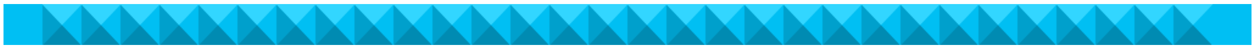
Depth 0

□ Persistent
☑ Clear Display Buffer
Width 1366
Height 768

```
DirectX11: Using hardware device
Collision Event time(microsecs)=3
Total memory used = 7703810(0x00758d02) bytes
*********************************.
Entering main loop.
*********************************.
```

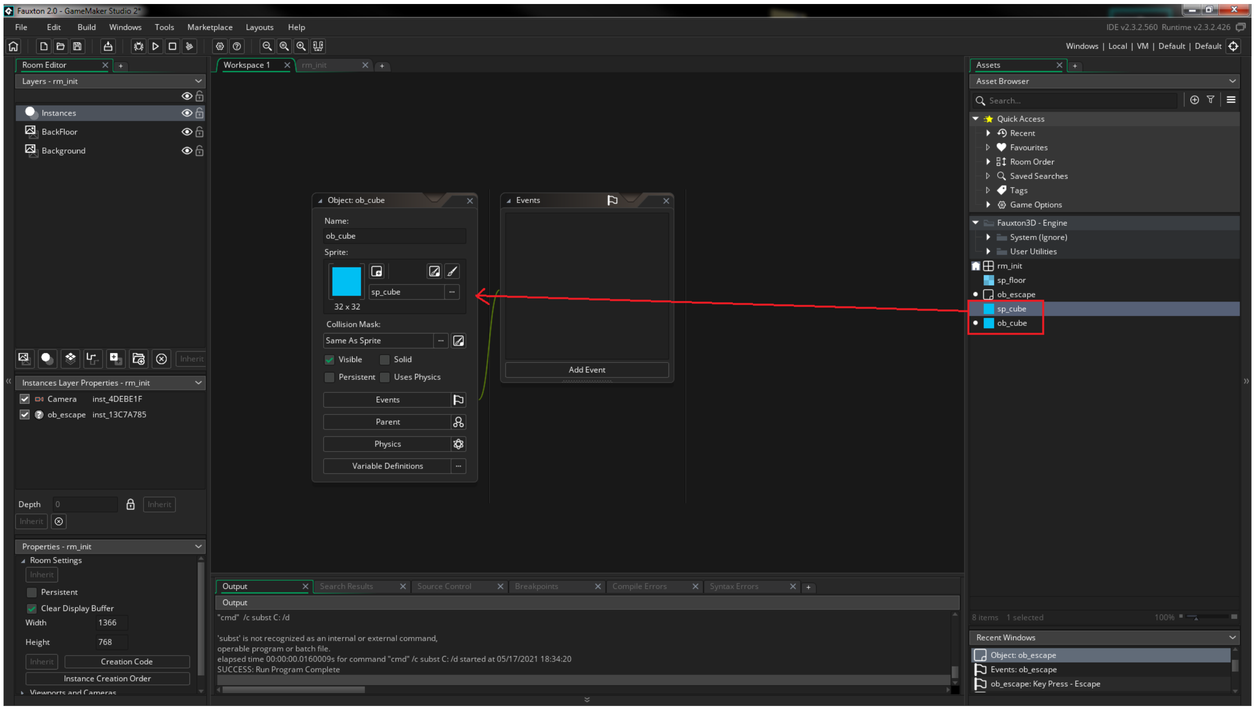
CREATING & UPDATING MODELS

Now, let’s create a model. First, we need to create a new 32x32 pixel sprite with 32 sub-images:



Next, let’s create a new object called **ob_cube** and set its sprite index to our new **sp_cube** sprite.

Fig 9.



Object: ob_cube

Name: ob_cube

Sprite: sp_cube (32 x 32)

Collision Mask: Same As Sprite

☑ Visible ☐ Solid

☐ Persistent ☐ Uses Physics

☑ Camera inst_4DEBE1F

☑ ob_escape inst_13C7A785

Depth 0

Room Settings: ☐ Persistent ☑ Clear Display Buffer, Width 1366, Height 768

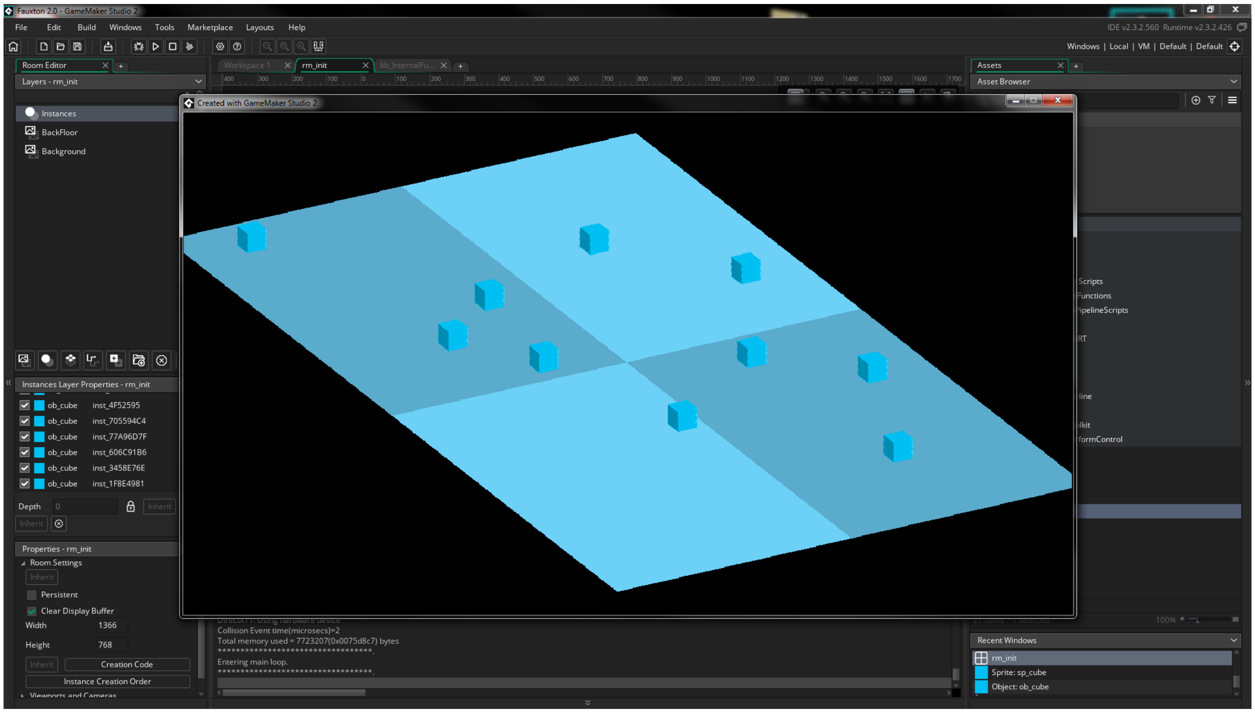
```
"cmd"  /c subst C: /d

'subst' is not recognized as an internal or external command,
operable program or batch file.
elapsed time 00:00:00.0160009s for command "cmd" /c subst C: /d started at 05/17/2021 18:34:20
SUCCESS: Run Program Complete
```

In the create event add this code:

```
model = fauxton_model_create(sprite_index, x, y, 0, 0, 0, 0, 1, 1, 1);
```

Then turn **ob_cube**'s visibility off and add a couple to the room! Now you should see something like this:

Fig 10.



Instances Layer Properties - rm_init

☑	ob_cube	inst_4F52595
☑	ob_cube	inst_705594C4
☑	ob_cube	inst_77A96D7F
☑	ob_cube	inst_606C91B6
☑	ob_cube	inst_3458E76E
☑	ob_cube	inst_1F8E4981

Depth 0

Properties - rm_init

Room Settings

□ Persistent

☑ Clear Display Buffer

Width 1366

Height 768

Creation Code

Instance Creation Order

Viewports and Cameras

```
Collision Event time(microsecs)=2
Total memory used = 7723207(0x0075d8c7) bytes
**********************************.
Entering main loop.
**********************************.
```

It’s that easy!! Now let’s make the objects move in circles! Add this to the step event of **ob_cube**

```
x += cos(current_time/250);
y += sin(current_time/250);

fauxton_model_set(model, x, y, 0, 0, 0, 0, 1, 1, 1);
```

If you run the game now you should see the boxes moving in little circles! Isn’t that easy! Play around, see what happens!

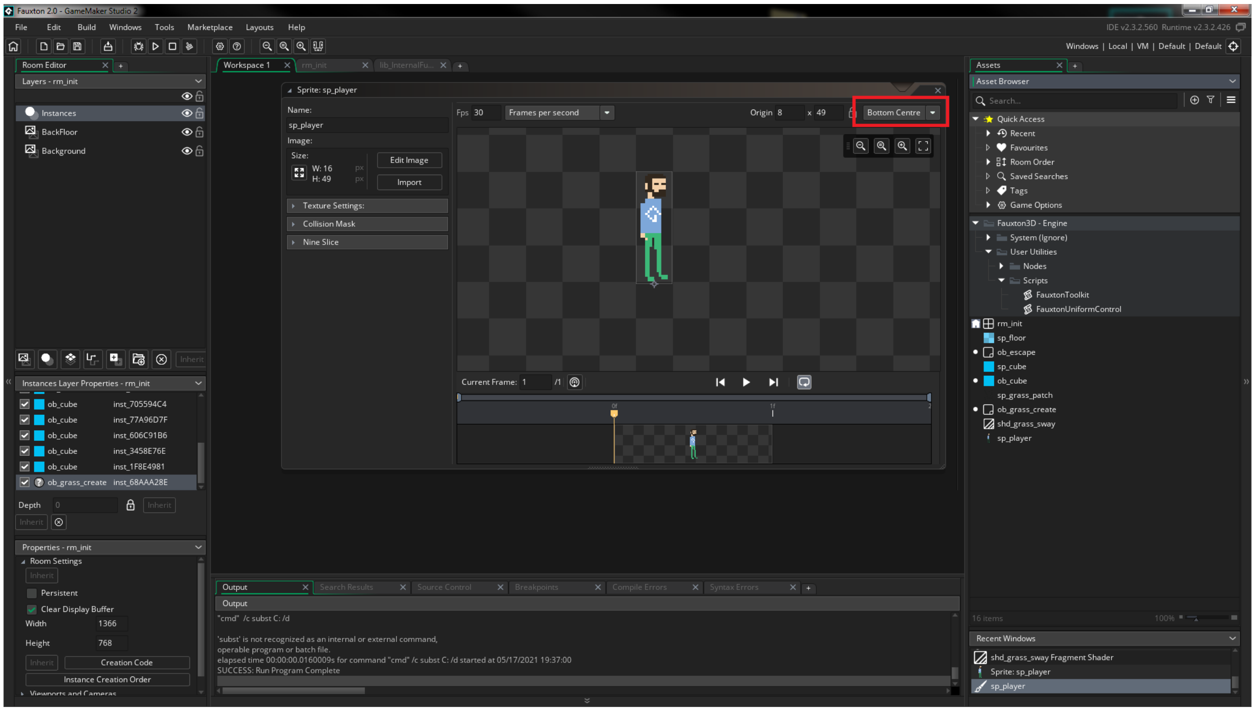
NOTE Currently the engine will give strange results when you modify the x scale and y scale of a model that includes rotation. This will hopefully be fixed in future updates, but for now, it’s best to model sprites as you would want them to appear un-modified and only rotate around the z-axis.

3D SPRITES

Now we have models but what about sprites? Well, this is made incredibly easy as well with the Sprite functions! For the most part, they work as you would expect, but now there is a SUPER easy way to make them face the camera at all times by just setting a **face_camera** parameter to true! As an example let's create a character and move them!
First, let's create a sprite and call it **sp_player**

Make sure to set the origin of the sprite to Bottom Centre.

Fig 11.



Sprite: sp_player

Name: sp_player

Size: W: 16 px H: 49 px

Fps 30 Frames per second

Origin 8 x 49 Bottom Centre

Current Frame: 1 /1

```
"cmd"  /c subst C: /d

'subst' is not recognized as an internal or external command,
operable program or batch file.
elapsed time 00:00:00.0160009s for command "cmd" /c subst C: /d started at 05/17/2021 19:37:00
SUCCESS: Run Program Complete
```

Now, let's create a new object with our sprite set as the object's sprite and call it **ob_player**. In the respective events add these:

Create Event

```
Camera.Target = id;
```

Step Event

```
var _h = keyboard_check(ord("D")) - keyboard_check(ord("A"));
var _v = keyboard_check(ord("S")) - keyboard_check(ord("W"));

var _moving = point_distance(0, 0, _h, _v) > 0.5;
var spd = 2;
var fric = 0.3;

if ( _moving )
{
      direction = point_direction(0, 0, _h, _v) + Camera.Forward;
}
speed = lerp(speed, _moving * spd, fric);
```

Draw Event

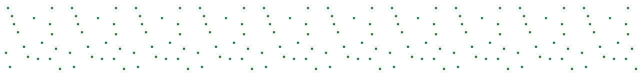
```
draw_sprite_3d(sprite_index, image_index, x, y, 0, 0, 0, 0, 1, 1, 1,
true);
```

Now if you add the player to the room the camera will follow us AND the player will billboard perfectly to the camera!

***NOTE** that the variable face_camera will override a sprites rotation and scale values so keep this in mind. If you want to rescale a sprites x scale, y scale, or rotation you should use the function **fauxton_sprite_set** and then draw your sprite using the regular *draw_sprite* functions. Make sure to **ALWAYS** set the x/y position of the draw_sprite function you are using to 0, 0 since the function **fauxton_sprite_set** will set the position of the sprite-based in the world, so you don't need to with Game Maker Studio 2's built-in *draw_sprite* x/y value.

STATIC BUFFERS & SHADERS

Static buffers are INCREDIBLY powerful and massive performance boosters. They are perfect for things that you do not intend to alter much. In this example, we will create some grass and make it sway!
First, let's import a simple 32x32 pixel grass patch sprite with 10 sub-images and call it **sp_grass_patch** :



Next, let's create an object called **ob_grass_create** and add this to the create event:

```
fauxton_buffer_create("GrassBuffer");

grassPatch = fauxton_model_create( sp_grass_patch, 0, 0, 0, 0, 0, 0,
1, 1, 1);
repeat(100)
{
      var xx, yy, rot, scl;
      xx = random(room_width);
      yy = random(room_height);
      rot = random(360);
      scl = random_range(0.5, 1.5);

      fauxton_model_set(grassPatch, xx, yy, 0, 0, 0, rot, scl, scl,
scl);
      fauxton_model_add_static(grassPatch, "GrassBuffer");
}
fauxton_model_destroy(grassPatch);
```

If you run the game now you will see a bunch of randomized grass! Not only is this insanely optimal for performance, but it is also perfect for something like grass, as we can also set custom shaders for buffers! Let’s create a new shader first and call it **shd_grass_sway**. In the vertex shader (*shd_grass_sway.vsh*) change the code to this:

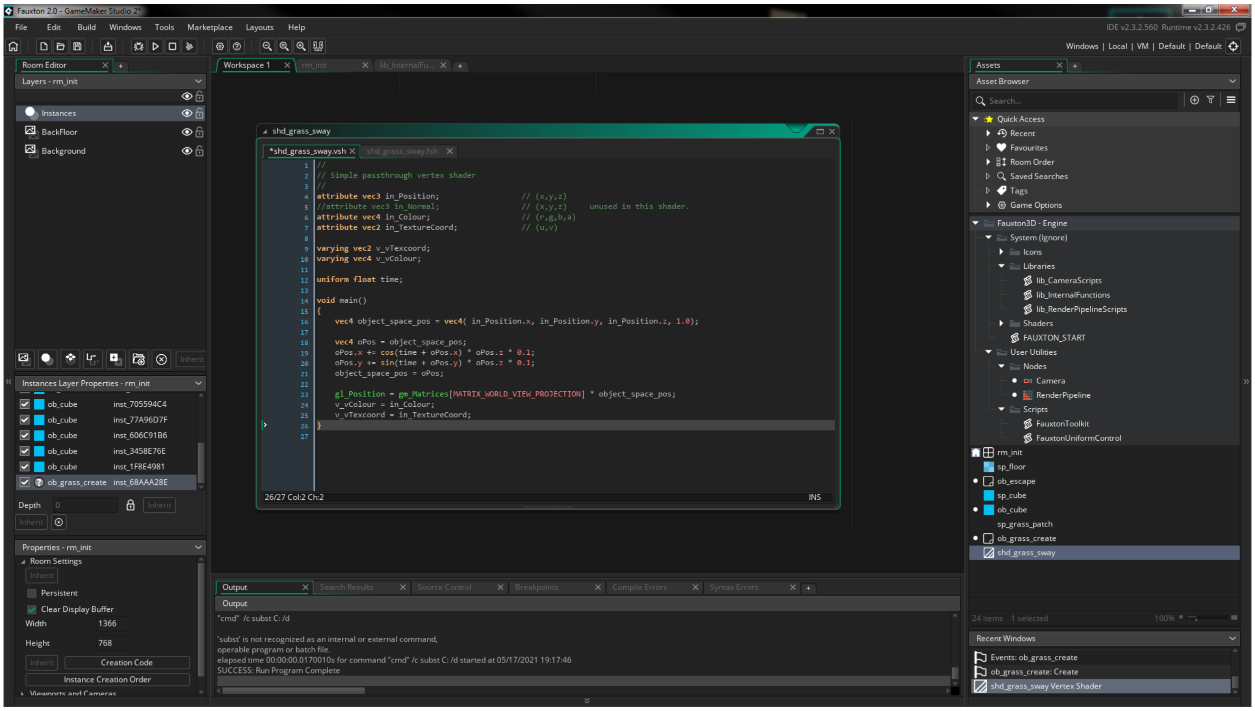
```
//
// Simple pass-through vertex shader
//
attribute vec3 in_Position;                  // (x,y,z)
//attribute vec3 in_Normal;                  // (x,y,z)     unused in
this shader.
attribute vec4 in_Colour;                    // (r,g,b,a)
attribute vec2 in_TextureCoord;              // (u,v)

varying vec2 v_vTexcoord;
varying vec4 v_vColour;

uniform float time;

void main()
{
    vec4 object_space_pos = vec4( in_Position.x, in_Position.y,
in_Position.z, 1.0);

      vec4 oPos = object_space_pos;
      oPos.x += cos(time + oPos.x) * oPos.z * 0.1;
      oPos.y += sin(time + oPos.y) * oPos.z * 0.1;
      object_space_pos = oPos;

    gl_Position = gm_Matrices[MATRIX_WORLD_VIEW_PROJECTION] *
object_space_pos;
    v_vColour = in_Colour;
    v_vTexcoord = in_TextureCoord;
}
```

And then change the fragment shader to this:

```
//
// Simple pass-through fragment shader
//
varying vec2 v_vTexcoord;
varying vec4 v_vColour;

void main()
{
        vec4 col = v_vColour * texture2D( gm_BaseTexture, v_vTexcoord );
        if ( col.a < 0.1 ) discard;

    gl_FragColor = col;
}
```

Since we are using custom shaders, Game Makers default alpha blending is overridden by your shader, so we will need to add our own in the fragment shader!

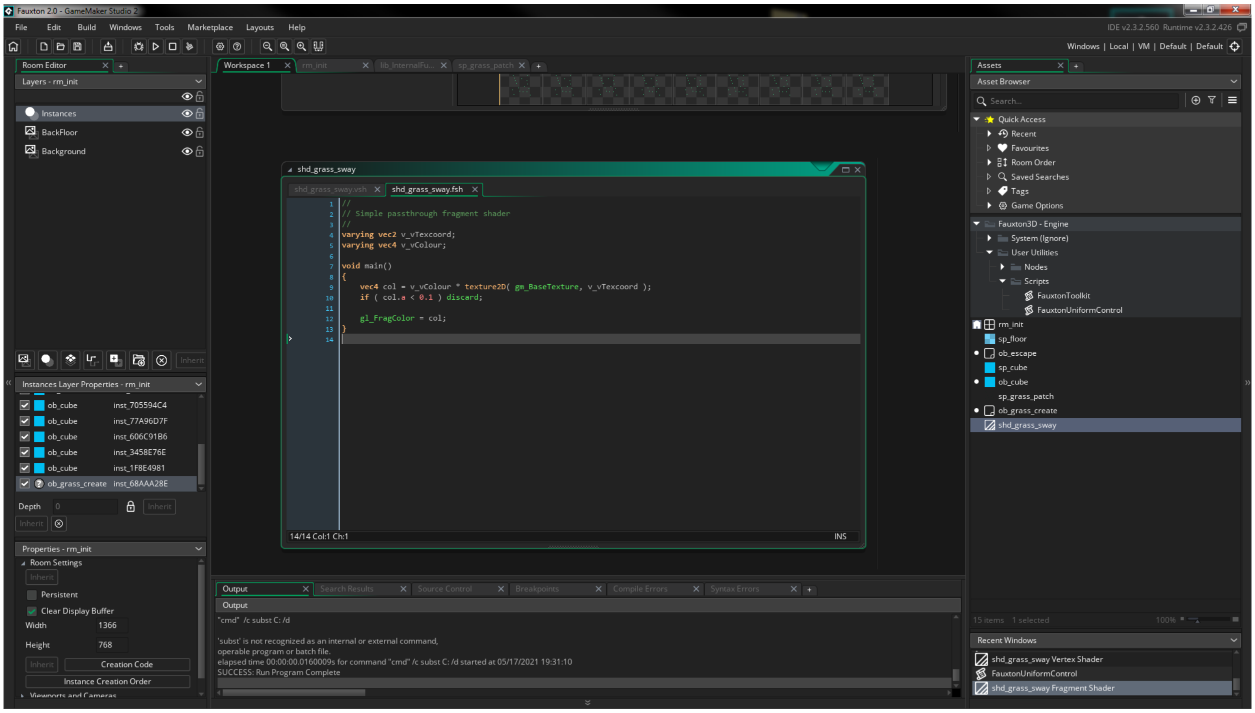
Fig 12.a



```
//
// Simple passthrough vertex shader
//
attribute vec3 in_Position;                  // (x,y,z)
//attribute vec3 in_Normal;                  // (x,y,z)     unused in this shader.
attribute vec4 in_Colour;                    // (r,g,b,a)
attribute vec2 in_TextureCoord;              // (u,v)

varying vec2 v_vTexcoord;
varying vec4 v_vColour;

uniform float time;

void main()
{
    vec4 object_space_pos = vec4( in_Position.x, in_Position.y, in_Position.z, 1.0);

    vec4 oPos = object_space_pos;
    oPos.x += cos(time + oPos.x) * oPos.z * 0.1;
    oPos.y += sin(time + oPos.y) * oPos.z * 0.1;
    object_space_pos = oPos;

    gl_Position = gm_Matrices[MATRIX_WORLD_VIEW_PROJECTION] * object_space_pos;
    v_vColour = in_Colour;
    v_vTexcoord = in_TextureCoord;
}
```

```
"cmd"  /c subst C: /d

'subst' is not recognized as an internal or external command,
operable program or batch file.
elapsed time 00:00:00.0170010s for command "cmd" /c subst C: /d started at 05/17/2021 19:17:46
SUCCESS: Run Program Complete
```

Fig 12.b



```
//
// Simple passthrough fragment shader
//
varying vec2 v_vTexcoord;
varying vec4 v_vColour;

void main()
{
    vec4 col = v_vColour * texture2D( gm_BaseTexture, v_vTexcoord );
    if ( col.a < 0.1 ) discard;

    gl_FragColor = col;
}
```

```
"cmd"  /c subst C: /d

'subst' is not recognized as an internal or external command,
operable program or batch file.
elapsed time 00:00:00.0160009s for command "cmd" /c subst C: /d started at 05/17/2021 19:31:10
SUCCESS: Run Program Complete
```

In this shader, we have a uniform called **time**. We will use this to gradually make our grass sway. But how do we set this to the grass buffer? Easy! If we go back into **ob_grass_create** we can add a new parameter to the end of our fauxton_buffer_create function:

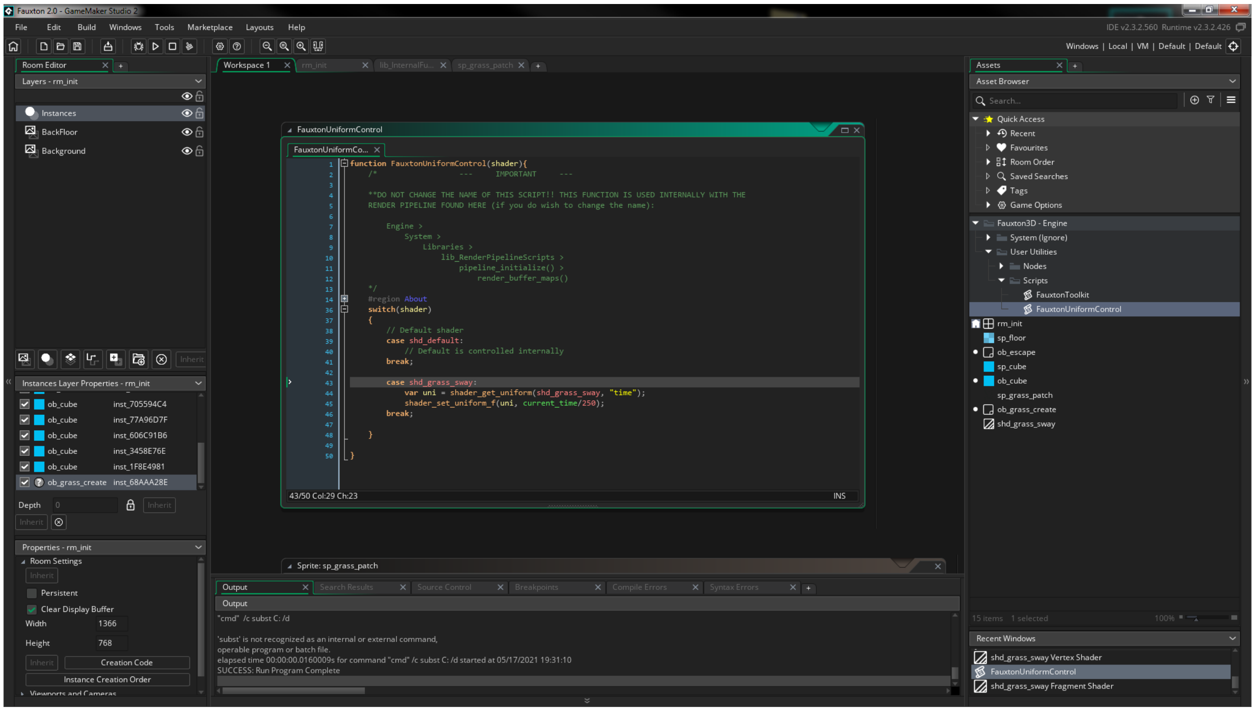
```
fauxton_buffer_create("GrassBuffer", shd_grass_sway);

grassPatch = fauxton_model_create( sp_grass_patch, 0, 0, 0, 0, 0, 0,
1, 1, 1);
repeat(100)
{
     var xx, yy, rot, scl;
     xx = random(room_width);
     yy = random(room_height);
     rot = random(360);
     scl = random_range(0.5, 1.5);

     fauxton_model_set(grassPatch, xx, yy, 0, 0, 0, rot, scl, scl,
scl);
     fauxton_model_add_static(grassPatch, "GrassBuffer");
}
fauxton_model_destroy(grassPatch);
```

So you're probably wondering how we edit the **time** uniform? This is made SUPER easy using the function fauxton_buffer_set_uniform_script. We can quickly set up a uniform script and add it to our buffer. Add the lines in **bold** to the script above:

```
var uniform_script = function(){
    var uni = shader_get_uniform(shd_grass_sway, "time");
    shader_set_uniform_f(uni, current_time/250);
}
fauxton_buffer_create("GrassBuffer", shd_grass_sway);
fauxton_buffer_set_uniform_script(“GrassBuffer, uniform_script);

grassPatch = fauxton_model_create( sp_grass_patch, 0, 0, 0, 0, 0, 0,
1, 1, 1);
repeat(100)
{
     var xx, yy, rot, scl;
     xx = random(room_width);
     yy = random(room_height);
     rot = random(360);
     scl = random_range(0.5, 1.5);

     fauxton_model_set(grassPatch, xx, yy, 0, 0, 0, rot, scl, scl,
scl);
     fauxton_model_add_static(grassPatch, "GrassBuffer");
}
```

```
fauxton_model_destroy(grassPatch);
```

Fig 13.



```
function FauxtonUniformControl(shader){
    /*              ---     IMPORTANT     ---

    **DO NOT CHANGE THE NAME OF THIS SCRIPT!! THIS FUNCTION IS USED INTERNALLY WITH THE
    RENDER PIPELINE FOUND HERE (if you do wish to change the name):

        Engine >
            System >
                Libraries >
                    lib_RenderPipelineScripts >
                        pipeline_initialize() >
                            render_buffer_maps()
    */
    #region About
    switch(shader)
    {
        // Default shader
        case shd_default:
            // Default is controlled internally
        break;

        case shd_grass_sway:
            var uni = shader_get_uniform(shd_grass_sway, "time");
            shader_set_uniform_f(uni, current_time/250);
        break;

    }

}
```

```
43/50 Col:29 Ch:23                                                    INS
```

```
"cmd"  /c subst C: /d

'subst' is not recognized as an internal or external command,
operable program or batch file.
elapsed time 00:00:00.0160009s for command "cmd" /c subst C: /d started at 05/17/2021 19:31:10
SUCCESS: Run Program Complete
```

Now if you run the game (and look closely) you can see the grass swaying! It's that easy!

LIGHTING

With Fauxton 3D you can easily add lighting to a scene by just adding an instance of **WorldEnvironment** and either point lights or spotlights! Fauxton supports up to **64** different spot/point lights. This can be changed, however, by going into **shd_default** and change all numbers in the *fragment* shader that are *64*. Let's look at what each of these nodes contains!

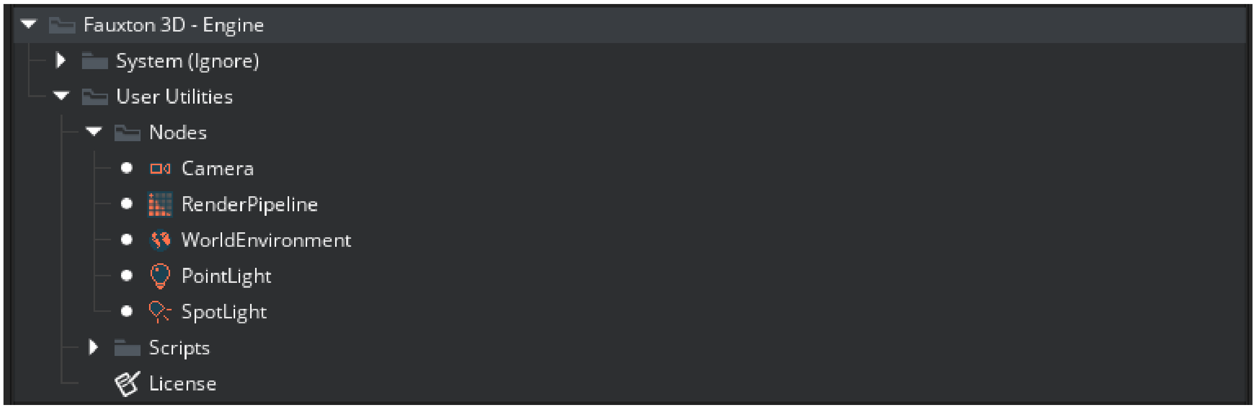
**** NOTE ****

If you override buffer shaders with your own you will have to calculate lighting in your new shader! It is suggested that you should duplicate the **shd_default** shader and its uniform script found in:

RenderPipeline
> *pipeline_initiate()*
>> *default_world_shader_set()*

And then proceed to make your changes.

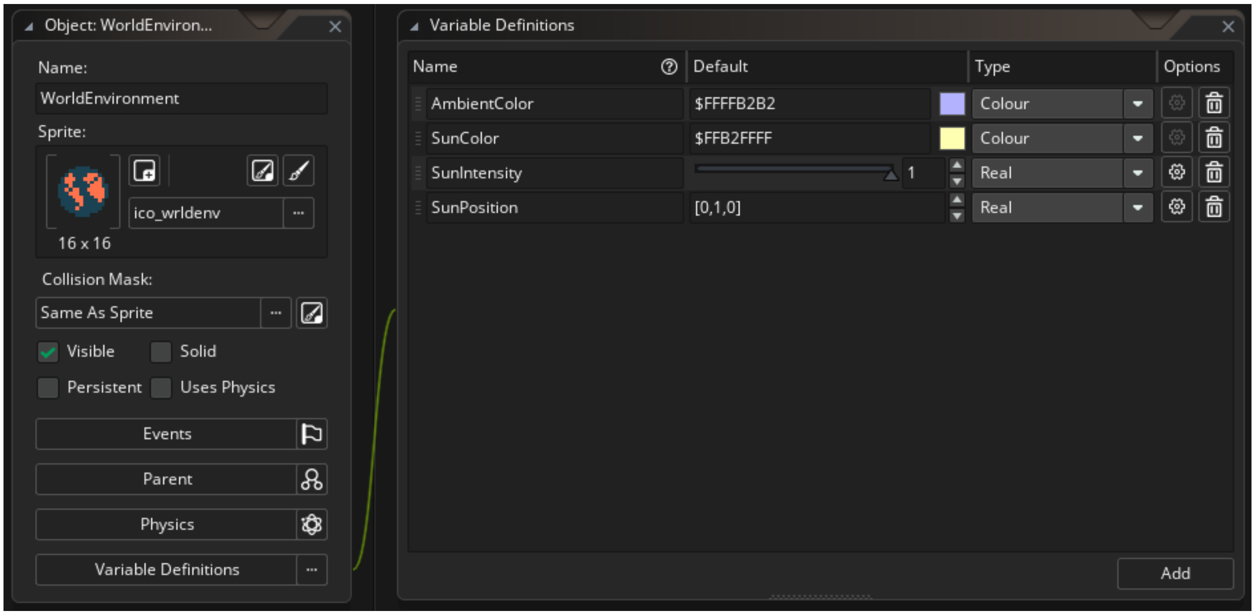
Fig 14



World Environment

First we have the **WorldEnvironment** node. This node MUST be added to a room in order to enable lighting. There are a few options in this node under the *Variable Definitions* button.

Fig 15



Name: WorldEnvironment

Sprite: ico_wrldenv

16 x 16

Collision Mask: Same As Sprite

☑ Visible ☐ Solid

☐ Persistent ☐ Uses Physics

Name	Default	Type	Options
AmbientColor	$FFFFB2B2	Colour	
SunColor	$FFB2FFFF	Colour	
SunIntensity	1	Real	
SunPosition	[0,1,0]	Real	

AmbientColor	The ambient color of our scene. The 'Shadow' color in a sense.
SunColor	The Color of our directional light
SunIntensity	The intensity of our directional light
SunPosition	The position of our directional light

The *AmbientColor* will be the color of the 'shadows' or rather the color of the scene facing away from the sun. Setting this to white ($ffffff / c_white) will make it so that the shadows are completely illuminated.

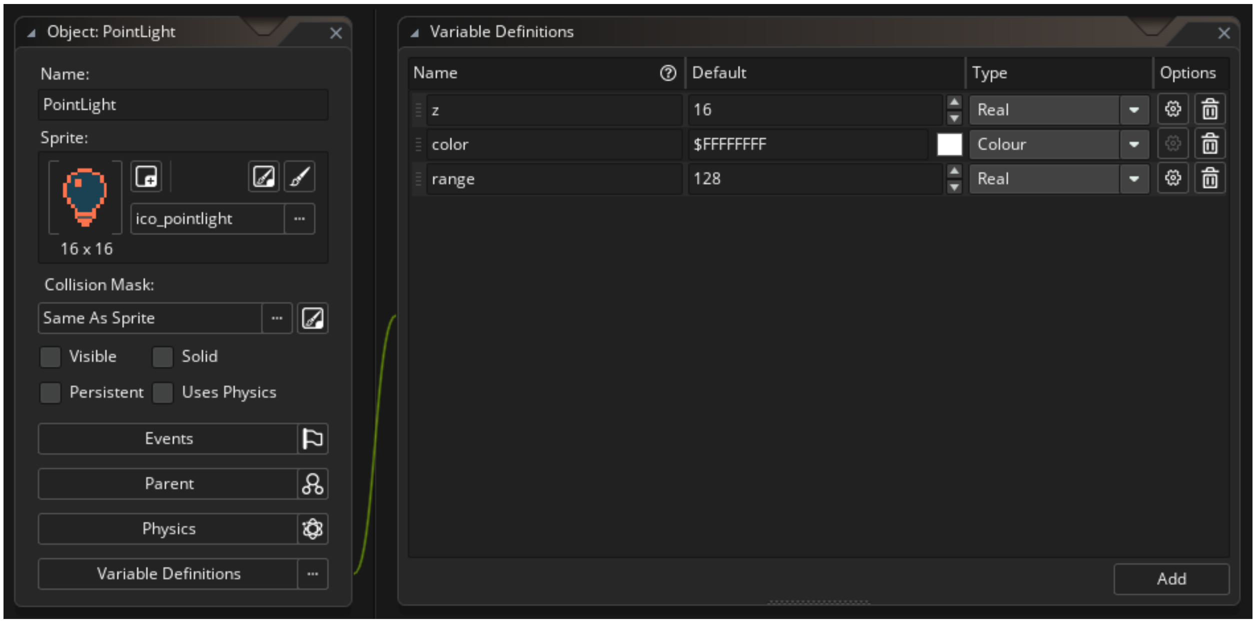
The SunColor is the color of our scene lit from the *SunPosition.* The sun position is an array that contains an X, Y, and Z. These values should be between a value of -1 and 1. So for example:

[-1, -1, 0]	The sun is at the **TOP-LEFT** corner of the room at a z of 0 (straight on)
[1, 1, 1]	The sun is at the **BOTTOM-RIGHT** corner of the point **DOWN** in z-space

Due to the fact that sprite-stacks are just a series of stacked planes on top of one another it should generally be avoided setting the sun's z-position to -1 (as you will not easily be able to see them illuminated from the bottom).

Point Lights

Point lights are SUPER easy to add in. Once you have an instance of **WorldEnvironment** added to your room you can add up to **64** point or spot lights! You can find the attributes for point lights in the *Variable Definitions*:



Object: PointLight

Name: PointLight

Sprite: ico_pointlight

16 x 16

Collision Mask: Same As Sprite

□ Visible □ Solid

□ Persistent □ Uses Physics

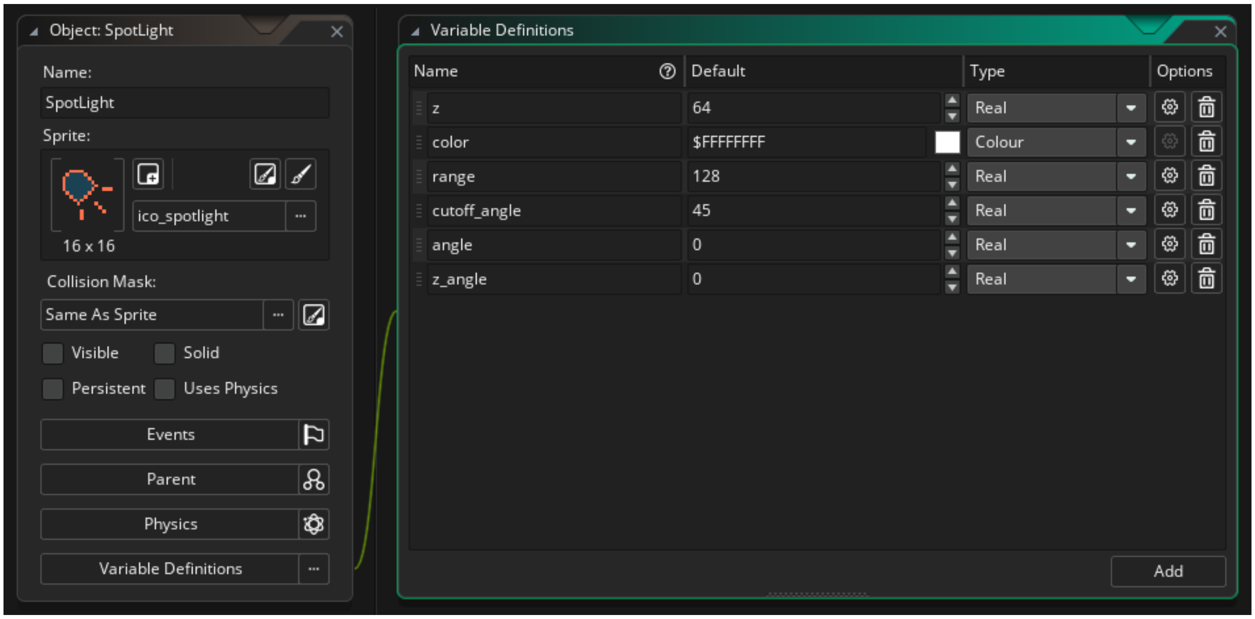
Variable Definitions

Name	Default	Type
z	16	Real
color	$FFFFFFFF	Colour
range	128	Real

z	The z position of the light
color	The Color of the light
range	The radius or 'range' of the light

Spot Lights

Spotlights are SUPER easy to add in. Once you have an instance of **WorldEnvironment** added to your room you can add up to **64** point or spot lights! You can find the attributes for spotlights in the *Variable Definitions*:



Object: SpotLight

Name: SpotLight

Sprite: ico_spotlight

16 x 16

Collision Mask: Same As Sprite

□ Visible □ Solid

□ Persistent □ Uses Physics

Variable Definitions

Name	Default	Type
z	64	Real
color	$FFFFFFFF	Colour
range	128	Real
cutoff_angle	45	Real
angle	0	Real
z_angle	0	Real

z	The z position of the light
color	The Color of the light
range	The radius or 'range' of the light
cutoff_angle	How many degrees to cut light to (smaller = spike, larger = bowl)
angle	The x/y angle of the light (imagine this like image_angle)
z_angle	The z angle of the light (how much the light points up or down)

FUNCTIONS

Model Functions

fauxton_model_create

This function will create a new sprite-stacked model using the sprite sub-images provided. Creating models using this function will add them to the render pipeline.

Syntax:

```
fauxton_model_create( sprite, subimg, x, y, z, xrot, yrot, zrot, xscale, yscale, zscale );
```

Argument	*Description*
sprite	The index of the sprite to model
x	The x coordinate of the model
y	The y coordinate of the model
z	The z coordinate of the model
xrot	The rotation around the x-axis of the model
yrot	The rotation around the y-axis of the model
zrot	The rotation around the z-axis of the model
xscale	The scale along the x-axis of the model
yscale	The scale along the y-axis of the model
zscale	The scale along the z-axis of the model

Returns:

Model ID (integer)

Example:

```
myModel = fauxton_model_create( spr_building, 100, 100, 0, 0, 0,
random(360), 1, 1, 1);
```

This will create a model at the room x coordinate 100 and room y coordinate 100 with a random rotation of 360 degrees.

fauxton_model_create_ext

This function will create a new sprite-stacked model using the sprite sub-images provided. Creating models using this function will add them to the render pipeline.

NOTE when dealing with alpha values in 3D you should always consider using the function faxuton_model_draw_override in the **draw_end** event as models with alpha values less than 1 drawn before models with alpha values at 1 will produce strange results.

Syntax:

```
fauxton_model_create_ext( sprite, subimg, x, y, z, xrot, yrot, zrot, xscale, yscale, zscale,
blend, alpha );
```

Argument	*Description*
sprite	The index of the sprite to model
x	The x coordinate of the model
y	The y coordinate of the model
z	The z coordinate of the model
xrot	The rotation around the x-axis of the model
yrot	The rotation around the y-axis of the model
zrot	The rotation around the z-axis of the model
xscale	The scale along the x-axis of the model
yscale	The scale along the y-axis of the model
zscale	The scale along the z-axis of the model
blend	The blend color of the model
alpha	The alpha value of each stack of the model

Returns:

Model ID (integer)

Example:

```
myModel = fauxton_model_create( spr_building, 100, 100, 0, 0, 0,
random(360), 1, 1, 1, c_blue, 0.5);
```

This will create a model at the room x coordinate 100 and room y coordinate 100, a random rotation of 360 degrees, the color blue for all layers, and an alpha of 0.5.

fauxton_model_texcube

This function will create a new sprite-stacked model using only 1 sprite or texture provided. Creating models using this function will add them to the render pipeline. **NOTE** models created using this function must be destroyed manually using fauxton_model_texcube_destroy!

Texcubes by default have a width/height of 1x1. You should use fauxton_model_set to change the scaling, rotation, and position of texcubes.

Syntax:

```
fauxton_model_texcube( height, texture );
```

Argument	*Description*
height	The height (layers) of the cube
texture	The texture of the cube

Returns:

Model ID (integer)

Example:

```
myModel = fauxton_model_texcube( 50, sprite_get_texture(spr_building,
0));
fauxton_model_set(myModel, x, y, 0, 0, 0, 0, 50, 50, 1);
```

This will create a texcube model with a height of 50 layers. We then set the texcube models position and x/y scale.

fauxton_model_texcube_destroy

This function will destroy a previously created texcube model (returned by the function fauxton_model_texcube)

Syntax:

```
fauxton_model_texcube_destroy( model_id );
```

Argument	*Description*
model_id	The index of the model to destroy

Returns:

N/A

Example:

```
myModel = fauxton_model_texcube( 50, sprite_get_texture(sp_building,
0));
fauxton_model_texcube_destroy(myModel);
```

This will create a texcube model and then instantly destroy that model.

fauxton_model_set

This function will set the position, rotation, and scale of a previously created sprite-stacked model (returned by the function fauxton_model_create)

Syntax:

```
fauxton_model_set( model_id, subimg, x, y, z, xrot, yrot, zrot, xscale, yscale, zscale );
```

Argument	*Description*
model_id	The index of the model to set
x	The x coordinate of the model
y	The y coordinate of the model
z	The z coordinate of the model
xrot	The rotation around the x-axis of the model
yrot	The rotation around the y-axis of the model
zrot	The rotation around the z-axis of the model
xscale	The scale along the x-axis of the model
yscale	The scale along the y-axis of the model
zscale	The scale along the z-axis of the model

Returns:

N/A

Example:

```
myModel = fauxton_model_create( spr_building, 0, 0, 0, 0, 0, 0 1, 1,
1);
fauxton_model_set( myModel, 100, 100, 0, 0, 0, random(360), 1, 1, 1);
```

This will create a model at the room x coordinate 0and room y coordinate 0 and then set the model to room x coordinate 100, room y coordinate 100, and z-rotation to a random rotation of 360 degrees.

fauxton_model_destroy

This function will destroy a previously created sprite-stacked model (returned by the function fauxton_model_create) and remove it from Fauxton's internal render queue.

Syntax:

```
fauxton_model_destroy( model_id );
```

Argument	*Description*
model_id	The index of the model to destroy

Returns:

N/A

Example:

```
myModel = fauxton_model_create( spr_building, 0, 0, 0, 0, 0, 0 1, 1,
1);
fauxton_model_destroy(myModel);
```

This will create a model and then instantly destroy that model.

fauxton_model_add_static

This function will add a previously created sprite-stacked model (returned by the function fauxton_model_create) to a previously created buffer

Syntax:

```
fauxton_model_add_static( model_id , buffer_name);
```

Argument	*Description*
model_id	The index of the model to add
buffer_name	The name of the static buffer you want to add a model to

Returns:

N/A

Example:

```
myModel = fauxton_model_create( spr_grass_patch, 0, 0, 0, 0, 0, 0 1,
1, 1);
fauxton_buffer_create("GrassBuffer");
fauxton_model_set(myModel, random(room_width), random(room_height), 0,
0, 0, random(360), 1, 1, 1);
fauxton_model_add_static(myModel, "GrassBuffer");
```

Here we first create a model, then we create a static buffer called "GrassBuffer" (using fauxton_buffer_create). We then set the model to a random room coordinate with a random rotation of 360 degrees. After we have set the new model's position, rotation, and scale we then add the model to our buffer.

fauxton_model_draw_enable

This function will enable or disable drawing of a previously created sprite-stacked model (returned by the function fauxton_model_create) by the engines internal render pipeline

Syntax:

```
fauxton_model_draw_enable( model_id, enabled );
```

Argument	*Description*
model_id	The index of the model to destroy
enable	Enable the model to be drawn by the engine (true/false)

Returns:

N/A

Example:

```
myModel = fauxton_model_create( spr_building, 0, 0, 0, 0, 0, 0 1, 1,
1);
fauxton_model_draw_enable(myModel, false);
```

This will create a model and then instantly set it so that it will not be automatically drawn by the engine.

fauxton_model_draw_override

This function will allow you to override the engine and draw a specific model yourself. This can be useful if you want to set a special shader or draw control for a specific model.

Syntax:

fauxton_model_draw_override(model_id);

Argument	*Description*
model_id	The index of the model to draw

Returns:

N/A

Example:

```
fauxton_model_draw_override( myModel );
```

This will override the engine’s internal drawing of a model that was previously created.

Sprite Functions

draw_sprite_3d

This function will allow you to draw 3D sprites.
***NOTE** setting *face_camera* to true will override any rotations and scales.

Syntax:

draw_sprite_3d(sprite, subimg, x, y, z, xrot, yrot, zrot, xscale, yscale, zscale, face_camera, *enable_lighting)

Argument	*Description*
sprite	The index of the sprite to draw
subimg	The sub-image of the sprite to draw
x	The x coordinate of the sprite
y	The y coordinate of the sprite
z	The z coordinate of the sprite
xrot	The rotation around the x-axis of the sprite
yrot	The rotation around the y-axis of the sprite
zrot	The rotation around the z-axis of the sprite
xscale	The scale along the x-axis of the sprite
yscale	The scale along the y-axis of the sprite
zscale	The scale along the z-axis of the sprite
face_camera	Set to always face camera (overrides rotation and scale)
enable_lighting	(optional) Allows 3D sprites blending with ambient and point lights

Returns:

N/A

Example:

```
draw_sprite_3d( sprite_index, image_index, x, y, 0, 0, 0, 0, 1, 1, 1,
true );
```

This will draw a sprite that always faces the camera.

draw_sprite_3d_ext

This function will allow you to draw 3D sprites.

***NOTE** setting *face_camera* to true will override any rotations and scales.

Syntax:

```
draw_sprite_3d_ext(sprite, subimg, x, y, z, xrot, yrot, zrot, xscale, yscale, zscale, blend, alpha, face_camera)
```

Argument	*Description*
sprite	The index of the sprite to draw
subimg	The sub-image of the sprite to draw
x	The x coordinate of the sprite
y	The y coordinate of the sprite
z	The z coordinate of the sprite
xrot	The rotation around the x-axis of the sprite
yrot	The rotation around the y-axis of the sprite
zrot	The rotation around the z-axis of the sprite
xscale	The scale along the x-axis of the sprite
yscale	The scale along the y-axis of the sprite
zscale	The scale along the z-axis of the sprite
blend	The color blend of the sprite
alpha	The alpha of the sprite
face_camera	Set to always face camera (overrides rotation and scale)
enable_lighting	(optional) Allows 3D sprites blending with ambient and point lights

Returns:

N/A

Example:

```
draw_sprite_3d_ext( sprite_index, image_index, x, y, 0, 0, 0, 0, 1, 1,
1, c_blue, 1, true );
```

This will draw a sprite that always faces the camera with a blended color of blue and an alpha of 1.

Buffer Functions

fauxton_buffer_create

This function allows you to create custom static buffers in Fauxton. You can set an optional variable ‘shader’ if you want to supply a custom shader. Shader uniforms should be set in the **FauxtonUniformControl** as a case in the provided switch statement.

Syntax:

```
fauxton_buffer_create( buffer_name, *shader );
```

Argument	*Description*
buffer_name	The name of the buffer (string)
shader	(Optional) shader of the buffer

Returns:

Buffer ID (index)

Example:

```
fauxton_buffer_create( “GrassBuffer”, shd_grass_sway );

grassModel = fauxton_model_create(spr_grass, 0, 0, 0, 0, 0, 0, 1, 1,
1);
repeat(100)
{
    var xx = random(room_width);
    var yy = random(room_height);
    var ang = random(360);
    var scl = random_range(0.5, 1.5);
    fauxton_model_set(grassModel, xx, yy, 0, 0, 0, ang, scl, scl,
scl);
    fauxton_model_add_static(grassModel, “GrassBuffer”);
}
fauxton_model_destroy(grassModel);
```

This will create a new custom buffer called “GrassBuffer” and set a custom shader for the buffer to *shd_grass_sway.* Then we create a new model and add 100 randomized instances of ‘grassModel’ to our static buffer. Once done we destroy our model (as it is no longer needed)

fauxton_buffer_get

This function allows you to retrieve the index of a previously created buffer by its name.

Syntax:

```
fauxton_buffer_get( buffer_name );
```

Argument	*Description*
buffer_name	The name of the buffer (string)

Returns:

Buffer ID (index)

Example:

```
fauxton_buffer_get( “GrassBuffer”);
```

This will return the buffer ID of a buffer named “GrassBuffer”.

fauxton_buffer_set

This function allows you to retrieve the index of a previously created buffer by its name.

Syntax:

```
fauxton_buffer_get( buffer_id_or_name, shader, matrix );
```

Argument	*Description*
buffer_name	The name (string) or ID (index) of the buffer to set
shader	The shader of the buffer to set
matrix	The new matrix of the buffer to set

Returns:

N/A

Example:

```
Var gBuff = fauxton_buffer_get( “GrassBuffer”);
fauxton_buffer_set( gBuff, shd_default, gBuff.matrix );
```

This will get the ID of the previously created buffer “GrassBuffer” and set its new shader to Fauxtons default shader, and set the matrix to the matrix it already has applied to it.

fauxton_buffer_set_uniform_script

This function allows you to set a custom shader uniform control script for a buffer created using *fauxton_buffer_create.*

Syntax:

```
fauxton_buffer_set_uniform_script( buffer_id_or_name, uniform_control_script);
```

Argument	*Description*
buffer_name	The name (string) or ID (index) of the buffer to set
uniform_control_script	The shader uniform control script for the buffer

Returns:

N/A

Example:

```
var uniform_script = function(){
    var uni = shader_get_uniform(shd_grass_sway, "time");
    shader_set_uniform_f(uni, current_time/250);
}
fauxton_buffer_create("GrassBuffer", shd_grass_sway);
fauxton_buffer_set_uniform_script(“GrassBuffer, uniform_script);
```

First we create a script to control our grass shader called **uniform_script**. We then create a buffer, set the shader to **shd_grass_sway** and set the uniform control script to our **uniform_script**.