Hi, welcome back to this last tutorial!

As said in the first tutorial (when we looked at the basics of draw event) the uiz\_cntn() function makes sure that stuff isn’t being drawn outside the area it’s supposed to be in. This brings some problems with it since the uiz contain methods have to use either shaders or surfaces (if surfaces aren’t supported). So for this reason when using either shaders or surfaces in your object outside of containment, there are certain things you need to do.

Let’s get started with surfaces. Normally you set the surface target, draw something and then RESET the surface using surface\_reset\_target(). To correctly utilize uiz at this point, you’ll have to replace surface\_reset\_target() with uiz\_contain\_regainsurface(). What this will do is set the surface to the original surface from uiz used to contain your object. If surfaces aren’t used as a containment method than surface\_reset\_target() will automatically be called. Also not that if you want to draw your surface after you have drawn something to it, you will have to do it AFTER calling uiz\_contain\_regainsurface(). Don’t use the regular surface\_reset\_target() on uiz objects.

Here is an example:

[CODE]

If uiz\_cntn() then{

//check surface

If !surface\_exists(surf) then{

//create surface as big as the object

Surf=surface\_create(width,height)

}

//set target and draw

Surface\_set\_target(surf)

Draw\_set\_color(c\_red)

Draw\_rectangle(rx,ry,rlx,rly,0)

Uiz\_contain\_regainsurface()

Draw\_surface(surf,rx,ry)

uiz\_containend()

}

[/CODE]

Normally you set the shader, draw something and then RESET the shader using shader\_reset(). Instead of doing that, you should use uiz\_contain\_regainshader(). No arguments are needed and this will automatically set or reset shaders. You don’t have to use shader\_reset().

Here is an example:

[CODE]

If uiz\_cntn() then{

//set shader and draw

Shader\_set(sh\_someshader)

Draw\_set\_color(c\_red)

Draw\_square(rx,ry,rlx,rly,0)

Uiz\_contain\_regainshader()

uiz\_containend()

}

[/CODE]

This allows you to use a shader, but now whenever you draw something with your shader the object isn’t contained because there isn’t any shader code being run in your shader. You have two options to fix this. The first is the easiest and that is to set “uiz\_shader=false” for that specific object which uses the shader. The other one is by embedding some of the uiz containment shader into your own shader. You can do this using the following steps: (see sh\_uiz\_contain for reference)

1. Create a vec4 called “c” or anything else if that variable name is already taken.
2. Create a uniform vec4 called “toy”.
3. Add the line “c = vec4(1.0,1.0,1.0,max(0.0,min(1.0,(1.0-min(1.0,floor(v\_pos.y/toy.a)))\*(1.0-min(1.0,floor(v\_pos.x/toy.z)))\*(floor(v\_pos.x/toy.x))\*(floor(v\_pos.y/toy.y)))));” to your fragment shader.
4. Multiply gl\_FragColor with “c” like: “*gl\_FragColor =* **c** *\* v\_vColour \* (texture2D( gm\_BaseTexture, v\_vTexcoord )+(1.0-ceil(v\_vTexcoord.x)\*ceil(v\_vTexcoord.y)));”*
5. Use shader\_get\_uniform to get a uniform id for “toy”. Put this code somewhere in create event or put it right before the next step:
6. Set the shader using shader\_set()
7. Now use something like the following code: “shader\_set\_uniform\_f(global.uniformid\_toy,cntnx,cntny,cntnlx,cntnly)”. Just replace global.uniformid\_toy with whatever you called your uniform id variable using shader\_get\_uniform.

All of this is done in the fragment shader or in gml, you don’t have to touch the vertex.

This should be everything to be able to utilize surfaces and shaders inside uiz objects. Now we’ll get into how to manipulate the uiz containment methods. To redefine the area in which uiz contains you can use uiz\_contain\_custom(x1,y1,x2,y2). This script uses pixel screen values. It could happen that you use this script and the custom containment area is so big that it will cause trouble drawing in places it shouldn’t be. For this reason you can use uiz\_contain\_custom\_min(x1,y1,x2,y2). This does almost the same as uiz\_contain\_custom but it has a “safety mechanism” that clamps your given area inside the maximum size before it starts interfering with other objects. If you only need to change containment area on one side you can use one of these four functions:

- uiz\_contain\_custom\_min\_right;

- uiz\_contain\_custom\_min\_left;

-uiz\_contain\_custom\_min\_top;

- uiz\_contain\_custom\_min\_bottom.

These functions all take on pixel argument corresponding to their side. Not that using one of these four functions will cancel each other out so using \_min\_left right after \_min\_top will

If you want to, I’d recommend looking inside obj\_uiZ\_loadingbar since it has some interesting mechanics it uses.

Draw event of obj\_uiZ\_loadingbar:

[CODE]

if uiz\_cntn() then{

shader\_set(sh\_uiz\_contain)

var p=round(rx+width\*uiz\_animation\_getfunction(value,animation)/2)

uiz\_contain\_custom\_min\_left(p)

uiz\_back(0)

uiz\_contain\_custom\_min\_right(p)

uiz\_back(2)

uiz\_containend()

}

[/CODE]

P is a pixel value on the screen (so not relative to the object). What we are trying to do this is to have to different backgrounds in our object, and have them not overlap. We first redefine our containment field to be shorter on the left, and after we’ve done that we draw our first background. After that we cut our containment area short on the right (canceling our cut on the left from 2 lines ago) and then drawing our second background.

I hope you can get working now with shaders and surfaces now. This also concludes everything you need to know to make your own objects. This last tutorial might have been a little bit more complicated than the rest because it’s also about more complicated game maker mechanics. If you don’t understand this tutorial very well, you shouldn’t worry that much since you probably won’t need to work with surfaces on shaders a whole lot for ui objects. But knowing how to make your own objects is still a good thing to know since your own stuff is still the most customizable stuff. Uiz is only a toolbox with a framework behind it in this regard. Most objects in uiz are based off customizability, and if you want an object that is a bit more static than you could always create one yourself.