Rendering In Orthographic Mode

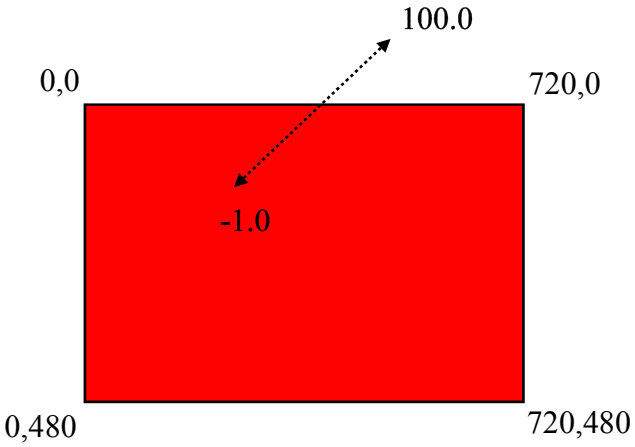
As a few of you are doing projects involving the rendering of 2D scenes, something of which I have some experience in (having written my own OpenGL GUI library), I thought it would be worth while writing about such rendering.

Orthographic Projection

Most of you should have at least some experience in using an orthographic projection of some sort - probably something that has the origin at the centre of the screen. That's quite useful for drawing centered quads and cubes for post-processing, but for a generic 2D renderer, we have a better option. We can create an orthographic projection that allows us to target the screen like a paint program's canvas, with the origin at the top left, and the extents of the X and Y axis at the bottom right. Using such a projection, it becomes very easy to accurately position objects directly using screen-space coordinates, rather than having to think about 3D world space.

It might at first be intuitive to have your screen's resolution dimensions as the extents, allowing simple pixel accurate positioning of screen objects. However, this brings about portability problems - if you use your screen-sized projection matrix on a screen with twice the resolution, your game will only take up the top left of the screen!

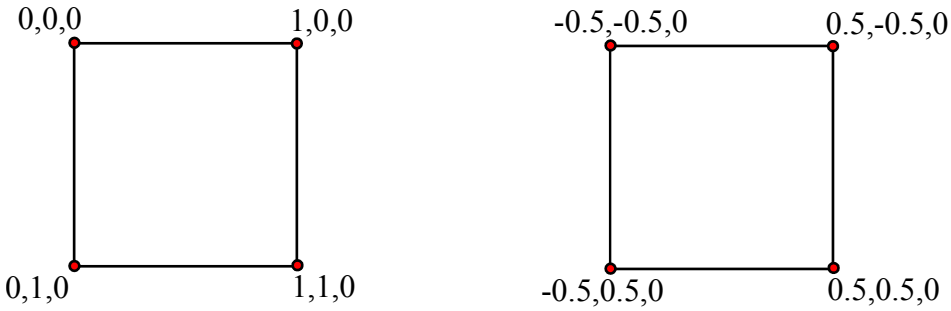
Instead, it is easier in the long run to pick an orthographic matrix that represents the ratio of the screen - for example, I use an orthographic matrix to create a 640x480 screen space for 4:3 ratio screens (old monitors, iPhone), and 720:480 for widescreen displays (most monitors, smartphones). This 'virtual' screen dimension will then cover the screen, while still providing intuitive, platform-agnostic screen units to work in.



Finally, it's worth mentioning the near and far planes of the orthographic matrix. For the far plane, any large number, say 100.0 or 1000.0, should suffice, while for the near plane, I generally use -1.0. I touched on why I do this in the graphics module notes, but I'll reiterate here. Generally, if you don't explicitly set the position of an object, whether it is projection on orthographic, it will be the origin, 0 0 0. Having a Z axis of 0 would mean the object would not show up in orthographic mode when using the 'common' near value of 1.0, and remember, you should never use a near value of 0 (it will cause a divide-by-zero)! A value of -1 will ensure that objects at the origin are drawn correctly, without risking a loss of precision.

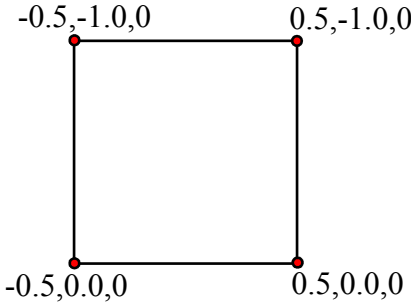
Drawing Objects

For drawing most things in orthographic mode, a textured quad should suffice. In my GUI renderer, I keep a single quad mesh, and use transformation matrices to translate it to the desired position, and to scale it to the desired size. To do this, you have a choice of two simple quads:



Both have sides of length 1 – this means you can scale them to a desired 'virtual pixel' size simply by using a scale matrix. So, to draw something on screen 400 units by 200 units, a scale value of 400,200,1 can be used. The only difference between the two quads is where their local origin is – the left quad has the origin at the top left, while the right has the origin at the centre. Which to use is entirely up to you – the left one is slightly better for object placement (placing the left quad at the origin will have the entire quad on screen, while the right would show only a quarter of its area), while the right-hand quad is slightly easier for rotation and scaling while staying in place.

If your game itself is in 2D, rather than just its user interface, you might want to consider the following, unit length quad, instead:



This would place the object centered and 'on top' of its position – handy for placing your game's sprite characters on screen.

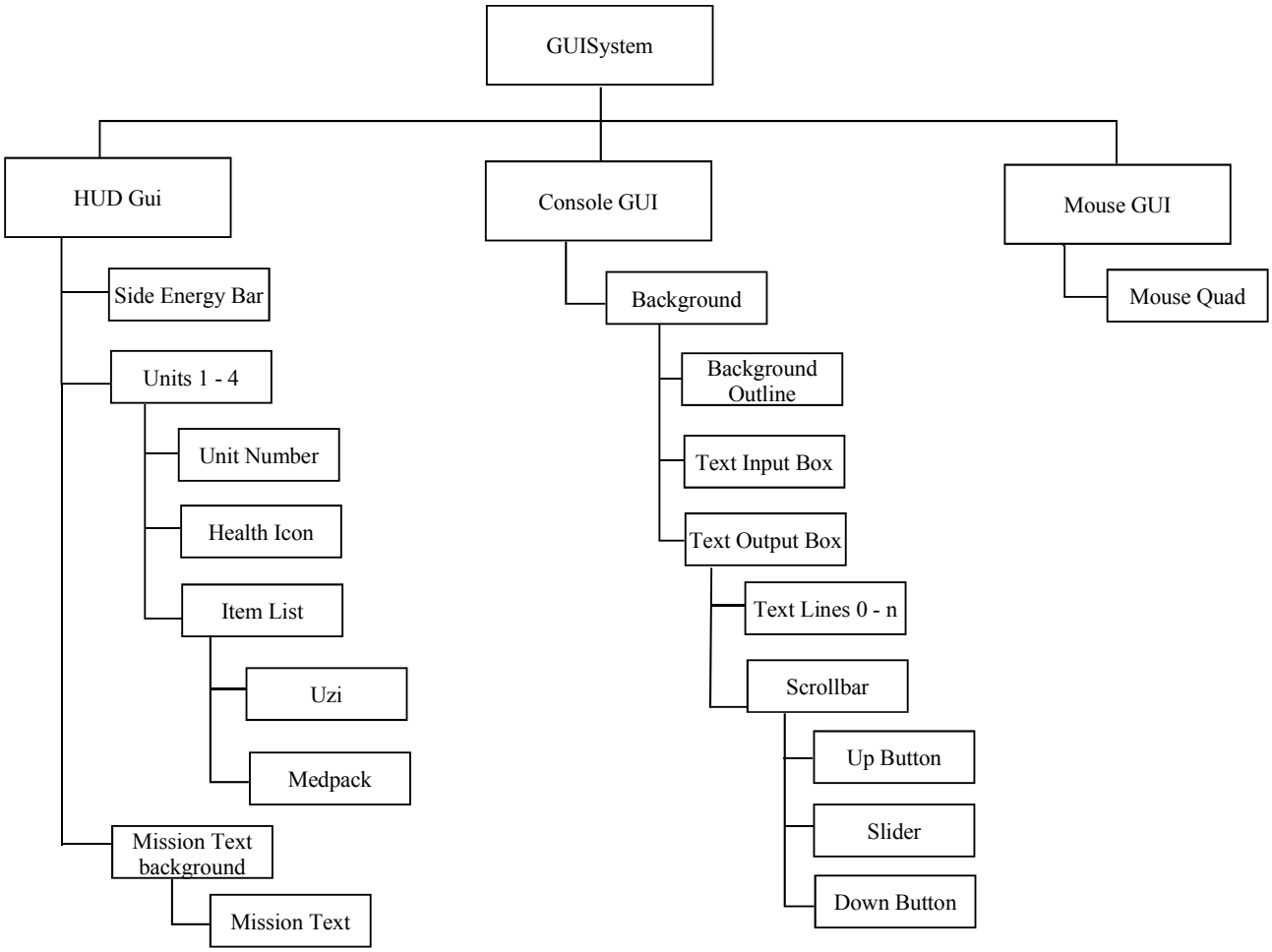
You might want your orthographic renderer to have access to all 3 quads, and choose between them as necessary – if you are careful when using these 'ortho quads', you can get away with switching the actively drawn VBO only a couple of times a frame, which is useful for maximising graphics cache efficiency.

Object Ordering

Maintaining a correct scene hierarchy is just as important when drawing orthographically as it is when drawing in perspective mode. In fact, with a correctly maintained ordering of orthographic nodes, you can disable depth testing and writing entirely, speeding up your rendering. Different parts of the graph structure may need to be drawn before others – for example child nodes should be drawn after their parent, and the mouse should always be on top of everything else, and so should be drawn last. This can easily be achieved by using the Z axis as a sorting criteria, either by using the depth buffer, or by sorting a list of nodes to draw by their z axis value – it is this latter option which allows the process of depth testing and writing to be skipped.

For example, here's a scene from my simple game, showing three orthographic elements – the game HUD, the debug console, and the mouse pointer. Each of these is a separate scene node hierarchy, and all subnodes are drawn relative to their parent, making it easy to move objects around.

These orthographic elements are part of my GUISystem, which sorts and renders orthographic objects by their Z axis order, with child nodes having a lower value than their parents. The scene node hierarchy of the above GUI scene is as follows:



As objects always have a lower Z value than their parents, a simple total draw ordering of nodes can be maintained by giving root nodes in my GUI different Z values – in this case the HUD has a Z value of 100, and the mouse GUI has a Z value of 0.0, ensuring the mouse is always drawn above everything else.

Object Clipping

There might be occasions where you wish to limit what is drawn to a particular area of the screen. For instance, in the GUI image on the previous page, the text rendering is limited to the area inside the purple box. Effects like this can be achieved using either the stencil buffer, or via a scissor region. 2D orthographic rendering is the ideal case for scissor regions, although remember that scissor regions are described in screen space, rather than in orthographic or clip space. In the GUI example, a parent node scissors out its region used while drawing its children, so the purple box scissors off the area used by the text, so strings do not 'bleed' out of their drawing region.