Overlaying on a 3D fullscreen application

I want to display some custom graphics on top of a 3rd party fullscreen Windows application.

Have you played any Steam games? It has an executable, GameOverlayUI.exe that lets you access Steam windows from within a game. (The GUI elements look custom-drawn, I don't know if that is a necessity; but they look exactly the same inside a game as they do on the desktop.) It even goes as far to 'dim' the game graphics in the background.

I'm wondering how one could go about writing such an application.

I'm also wondering how broad the scope of solutions would be. Could you use one method for all OpenGL applications? For all Direct3D minus DirectDraw applications? For all fullscreen Windows applications?

I'm looking for more 'modern' and generic solutions - overlaying a command prompt or a 320x240 indexed color application isn't a concern.

Edit: Some clarification: The fullscreen application isn't mine. It can be anything. Most probably, it will be a 3D game. I want to display, say a digital clock in the bottom right corner of the screen, on top of the game graphics. I would like to avoid tricks such as injecting code or debugging the process, if possible.

windows opengl directx overlay fullscreen

edited Dec 5 '10 at 7:35
Lodle
9,848 13 50 82

asked May 29 '09 at 13:32
aib
27.6k 10 56 73

How far did you manage to go with the above approach. I'm stuck in a similar situation and I want to draw chromium embedded webviews as overlay. Do you think that's possible? – alDiablo Apr 3 at 10:04

5 Answers

Solution

Well, here's another example. Xfire is a chat program that overlays its interface into games so you can receive messages while playing. The way they do this is by editing the d3d/opengl DLL at the process memory level, like injecting assembly jumps. I know this because their method was causing my mod to crash, and I actually saw the strange assembly code they were injecting into the d3d9.dll.

So here's how Xfire does this:

1. target the process of the game
2. search its process memory for d3d.dll/opengl.dll
3. inject into the process a DLL containing the custom interface logic
4. connect interface's functions to the program flow by manually writing assembly jumps in the d3d/opengl functions found in the process memory

There's no other conceivable way since you need to hijack the graphics device that belongs to that game. I'm willing to bet that Steam does it the same way as Xfire. So yeah, no easy way to do this unless someone created a helpful library to do everything you need to do at the low level.

You also asked about dimming the game graphics under your overlay. This is just a simple `DrawPrimitive` call to draw a filled transparent rectangle over the screen.

Created a Gist with full code here

This is a pretty esoteric art, and there are several ways to do this. I prefer what's called a *Direct3D Wrapper*. It's been years since I've done this, but I did it with a game once. I may have left out some details, but I just hope to illustrate the idea.

All Direct3D9 games need to call IDirect3DDevice9::EndScene after the screen is rendered and ready to be drawn. So in theory, we can put custom code inside `EndScene` that draws what we want over the game. We do this by creating a class that looks like IDirect3DDevice9 to the game, but it's really just a wrapper that forwards all function calls to the real class. So now, in our custom class, our wrapper function for `EndScene` looks like this:

```
HRESULT IDirect3DDevice9::EndScene()
{
   // draw overlays here

   realDevice.EndScene();
}
```

Then we compile it into a DLL called d3d9.dll, and drop it in the game executable's directory. The real d3d9.dll should be in the /system32 folder, but the game first looks in the current directory, and instead loads ours. Once the game loads, it uses our DLL to create an instance of our wrapper class. And now each time `EndScene` is called, our code is executed to draw what we want to the screen.

I think you can try the same principle with OpenGL. But to prevent tampering, I think some modern games try to prevent this.

answered Nov 5 '09 at 20:54
Shaun LeBron
1,390 13 17

excellent. Coupled with your other answer, this is what I've been looking for. Thanks. – aib Nov 9 '09 at 7:30

9 Be careful when doing this with multiplayer games requiring anti cheat software like Punkbuster! Injecting custom libraries into running games is considered as intrusive game manipulation and cheating. Programs (like Xfire, Teamspeak Overlay, ...) are white-listed and won't cause cheating violations anymore but it's a learning process done by the Anti-Cheat-Vendor which can be time-consuming where your program can't be effectively used. – Robert Mar 30 '10 at 14:28

---

therers a free software project its name it's taksi link text. i'm not sure how this application override the process cause i'm be examining the code yet but checkit i think this is a good project

answered May 6 '10 at 15:52
community wiki
Car

I've been thinking about this. For OpenGL, I'd hook WGL's SwapBuffers with Detours, drawing my stuff after the game and then swapping buffers myself.

answered Nov 16 '09 at 18:11
user212277

I've done overlays with both DirectX & WPF viewports (same thing, really), using the overlay to paint some nice 2D GDI+ stuff on top of 3D visualisations.

I managed by using a panel on top of the "actual" visualisation, setting the colour to transparent except for the bits I wanted overlaid. Worked pretty well, but it was a bit of a pain passing click, drag, and other events through the "drawing" layer to the 3D control.

answered May 29 '09 at 13:39
moobaa
4,017 1 21 31

I'm sorry if my question wasn't clear: The "actual" visualisation is drawn by a 3rd party app, in my case. I've done what you said, once, in the context of a single application. I don't know if your answer is relevant in the context of two different applications; if it is, how would I go about creating a child panel in the context of the fullscreen parent? – aib May 30 '09 at 21:23