Rendering graphics in C#

Asked 16 years, 3 months ago Modified 8 years, 1 month ago

Viewed 21k times

9

Is there another way to render graphics in C# beyond GDI+ and XNA?

(For the development of a tile map editor.)

c# gdi+ xna rendering

Share

Improve this question

Follow

edited Nov 5, 2016 at 22:59

Peter Mortensen

31.6k • 22 • 109 • 133

asked Sep 12, 2008 at 2:56

David McGraw

5,177 • 7 • 37 • 36

David, just noticed your question "Understanding Pointers". Great question, great answers! All points mentioned in that post also apply to my "unsafe" answer below. – Ash Sep 12, 2008 at 3:34

6 Answers

Sorted by: Highest score (default)

10

SDL.NET is the solution I've come to love. If you need 3D on top of it, you can use Tao.OpenGL to render inside it. It's fast, industry standard (SDL, that is), and cross-platform.

Share Improve this answer

Follow

edited Nov 5, 2016 at 23:01

Peter Mortensen

31.6k ● 22 ● 109 ● 133

answered Sep 12, 2008 at 2:59

Serafina Brocious

30.6k ● 12 ● 91 ● 115

Awesome! I'm actually familiar with SDL. I'm going to look into this. – David McGraw Sep 12, 2008 at 3:04

They have moved their wiki so that link no longer works. Just goto cs-sdl.sourceforge.net and navigate from there – Danny Parker Nov 22, 2011 at 14:30

4

Yes, I have written a Windows Forms control that wraps DirectX 9.0 and provides direct pixel level manipulation of the video surface.

I actually wrote another post on Stack Overflow asking if there are other better approaches: *Unsafe C# and pointers for 2D rendering, good or bad?*

While it is relatively high performance, it requires the unsafe compiler option as it uses pointers to access the memory efficiently. Hence the reason for this earlier post.

This is a high level of the required steps:

1. Download the DirectX SDK.
2. Create a new C# Windows Forms project and reference the installed Microsoft DirectX assembly.
3. Initialize a new DirectX Device object with Presentation Parameters (windowed, back buffering, etc.) you require.
4. Create the Device, taking care to record the surface "Pitch" and current display mode (bits per pixel).
5. When you need to display something, `Lock` the backbuffer surface and store the returned pointer to the start of surface memory.
6. Use pointer arithmetic, calculate the actual pixel position in the data based on the surface pitch, bits per pixel and the actual x/y pixel coordinate.
7. In my case for simplicity I am sticking to 32 bpp, meaning setting a pixel is as simple as: * (surfacePointer + (y * pitch + x))=Color.FromARGB(255,0,0);
8. When finished drawing, `Unlock` the back buffer surface. Present the surface.
9. Repeat from step 5 as required.

Be aware that taking this approach you need to be very careful about checking the current display mode (pitch and bits per pxiel) of the target surface. Also you will need

to have a strategy in place to deal with window resizing or changes of screen format while your program is running.

Share Improve this answer

Follow

edited May 23, 2017 at 11:48

Community Bot

1 ● 1

answered Sep 12, 2008 at 3:02

Ash

62.1k ● 31 ● 155 ● 172

2

- Managed DirectX (Microsoft.DirectX namespace) for faster 3D graphics. It's a solid .NET wrapper over DirectX API, which comes with a bit of performance hit for creating .NET objects and marshalling. Unless you are writing a full featured modern 3D engine, it will work fine.
- Window Presentation Foundation (WPF) (Windows.Media namespace) - best choice for 2D graphics. Also has limited 3D abilities. Aimed to replace Windows Forms with vector, hardware accelerated resolution-independent framework. Very convenient, supports several flavours of custom controls, resources, data binding, events and commands... also has a few WTFs. Speed is usually faster than GDI and slower than DirectX, and depends greatly on how you do things (seen something to work 60 times faster after rewriting in a sensible way). We had a success implementing 3

1280x1024 screens full of real-time indicators, graphs and plots on a single (and not the best) PC.

Share Improve this answer

Follow

answered Sep 12, 2008 at 4:33

ima

8,255 ● 3 ● 23 ● 19

2

You might look into the Cairo graphics library. The Mono project has bindings for C#.

Share Improve this answer

Follow

edited Nov 5, 2016 at 22:56

Peter Mortensen

31.6k ● 22 ● 109 ● 133

answered Nov 13, 2008 at 13:09

AngelBlaZe

464 ● 5 ● 11

1

You could try looking into WPF, using Visual Studio and/or Expression Blend. I'm not sure how sophisticated you're trying to get, but it should be able to handle a simple editor. Check out this MSDN Article for more info.

Share Improve this answer

Follow

answered Sep 12, 2008 at 3:21

tbreffni

5,132 ● 5 ● 32 ● 30

Cairo is an option. I'm currently rewriting my mapping software using both GDI+ and Cairo. It has a tile map

1

generator, among other features.

Share Improve this answer

Follow

edited Nov 5, 2016 at 22:56

Peter Mortensen

31.6k ● 22 ● 109 ● 133

answered Mar 30, 2009 at 13:37

Igor Brejc

19k ● 13 ● 79 ● 95