# What are OpenGL extensions, and what are the benefits/tradeoffs of using them?

Asked 16 years, 4 months ago  Modified 10 years ago  Viewed 5k times

17

In relation to this question on Using OpenGL extensions, what's the purpose of these extension functions? Why would I want to use them? Further, are there any tradeoffs or gotchas associated with using them?

opengl

Share

Improve this question

Follow

edited May 23, 2017 at 12:33

Community Bot

1 • 1

asked Aug 20, 2008 at 2:40

Rob Thomas

686 • 7 • 17

## 3 Answers

Sorted by: Highest score (default)

24

The OpenGL standard allows individual vendors to provide additional functionality through extensions as new technology is created. Extensions may introduce new

functions and new constants, and may relax or remove restrictions on existing OpenGL functions.

Each vendor has an alphabetic abbreviation that is used in naming their new functions and constants. For example, NVIDIA's abbreviation (NV) is used in defining their proprietary function glCombinerParameterfvNV() and their constant GL_NORMAL_MAP_NV.

It may happen that more than one vendor agrees to implement the same extended functionality. In that case, the abbreviation EXT is used. It may further happen that the Architecture Review Board "blesses" the extension. It then becomes known as a standard extension, and the abbreviation ARB is used. The first ARB extension was GL_ARB_multitexture, introduced in version 1.2.1. Following the official extension promotion path, multitexturing is no longer an optionally implemented ARB extension, but has been a part of the OpenGL core API since version 1.3.

Before using an extension a program must first determine its availability, and then obtain pointers to any new functions the extension defines. The mechanism for doing this is platform-specific and libraries such as GLEW and GLEE exist to simplify the process.



answered Aug 20, 2008 at 3:43

Andrew Grant

58.8k ● 22 ● 131 ● 144

Extensions are, in general, a way for graphics card vendors to add new functionality to OpenGL without having to wait until the next revision of the OpenGL spec. There are different types of extensions:

1. Vendor extension - only one vendor provides a certain type of functionality.
    - Example: `NV_vertex_program`
2. Multivendor extension - multiple vendors have gotten together and agreed on the functionality.
    - Example: `EXT_vertex_program`
3. ARB extension - the OpenGL Architecture Review Board has blessed the extension. You have a reasonable expectation that this type of extension will be around for a while.
    - Example: `ARB_vertex_program`

Extensions don't have to go through all of these steps. Sometimes an extension is only ever implemented by one vendor, before hardware designs go a different way and the extension is abandoned. Other times, an extension might make it as far as ARB status before everyone decides there's a better way. (The `ARB_vertex_program` approach, for instance, was set aside in favor of the high-level shading language approach of `ARB_vertex_shader` when it came time to roll shaders into the core OpenGL spec.) Even ARB extensions don't last forever; I wouldn't write something today requiring `ARB_matrix_palette`, for instance.

All of that having been said, it's a very good idea to keep up to date on extensions, in particular the latest ARB and EXT extensions. In the past it has been true that some of the 'fast paths' through the hardware were only accessible via extensions. Likewise, if you want to know what all functionality a piece of hardware can do, there's no better place to look than in a vendor-specific extension.

If you're just getting started in OpenGL, I'd recommend investigating:

- `ARB_vertex_buffer_object` (vertices)
- `ARB_vertex_shader` / `ARB_fragment_shader` / `ARB_shader_objects` / GLSL spec (shaders)

More advanced:

- `ARB/EXT_framebuffer_object` (off-screen rendering)

This is all functionality that's been rolled into core, but it can be good to see it in isolation so you can get a better feel for where its boundaries lie. (The core OpenGL spec seamlessly mixes the old with the new, so this can be pretty important if you want to stay on the fast path, and avoid the legacy and sometimes implemented in software paths.)

Whatever you do, make sure you have appropriate checks for the extensions you decide to use, and fallbacks where necessary. Even though your card may have a given extension, there's no guarantee that the

extension will be present on another vendor's card, or even on another operating system with the same card.

Share Improve this answer Follow

answered Aug 23, 2008 at 3:00

spate
386 ● 2 ● 5

---

7

**OpenGL Extensions** are new features added to the **OpenGL specification**, they are added by the OpenGL standards body and by the various graphics card vendors. These are exposed to the programmer as new function calls or variables. Every new version of the OpenGL specification ships with newer functionality and (typically) includes all the previous functionality and extensions.

The real problem with OpenGL extensions exists only on Windows. Microsoft hasn't supported any extensions that have been released after *OpenGL v1.1*. The graphics card vendors overcome this by shipping their own version of this functionality through header files and libraries. However, using this can be a bit painful as the question you linked to shows. But this problem has mostly gone away with the popularity of **GLEW**, which takes care of wrapping all this into a easy-to-use package.

If you do use a very recent OpenGL extension, be aware that it may not be supported on older graphics hardware. Other than this, there's no other disadvantage to using these extensions. Most of the extensions which become

standard are pretty darn *useful* and there's very little logic to not use them.

Share Improve this answer Follow

edited Aug 20, 2008 at 10:48

answered Aug 20, 2008 at 3:51

Ashwin Nanjappa
**78.3k** ● 89 ● 220 ● 296