## horizontal line



OpenGL

Σημειώσεις ΙΕΚ

ΤΕΧΝΙΚΟΣ ΕΦΑΡΜΟΓΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

Πολυμέσα

Web Designer - Developer

Video Games

# Επισκόπηση

Οι σημειώσεις αφορούν το μάθημα Γλώσσα Προγραμματισμού IV (OpenGL) του Γ’ Εξαμήνου ΙΕΚ στην ειδικότητα ΤΕΧΝΙΚΟΣ ΕΦΑΜΟΓΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ.

Έχουν σκοπό να τοποθετήσουν σε τάξη την (ευμεγέθη) πληροφορία που δέχεσαι καθημερινά ώστε αυτή να γίνει γνώση. Είναι χρήσιμες μόνο για καταρτιζόμενους που έχουν ανάγκη την δομημένη πληροφορία προκειμένου να την οικειοποιηθούν.

# Αξιοσημείωτα

1. Οι σημειώσεις δεν είναι “όλα όσα έχεις να μάθεις” προκειμένου να κατέχεις το γνωστικό αντικείμενο OpenGL. Είναι βοήθημα στη δική σου προσπάθεια για εξέλιξη πάνω στο γνωστικό αντικείμενο.
2. Περιέχουν πληθώρα υπερσυνδέσμων σε επιλεγμένο υλικό.
3. Είναι δίγλωσσες (ελληνικά και αγγλικά αναμεμιγμένα)

# Τι είναι η OpenGL;

Application programming interface (API) για να παρασταθούν δισδιάστατα 2D και τρισδιάστατα 3D διανυσματικά γραφικά (vector graphics).

Direct3D and OpenGL are competing application programming interfaces (APIs) which can be used in applications to render 2D and 3D computer graphics.

Direct3D application development targets the Microsoft Windows platform.

# Είναι η OpenGL το μοναδικό API για 2D & 3D rendering;

The OpenGL API is an open standard, which means that various hardware makers and operating system developers can freely create an OpenGL implementation as part of their system. OpenGL implementations exist for a wide variety of platforms. Most notably, OpenGL is the dominating graphics API of Unix-like computer systems.

**From an application developer's perspective, Direct3D and OpenGL are equally open**. Full documentation and necessary development tools are available with no restrictions.

In more detail, the two computer graphics APIs are the following:

**Direct3D is a proprietary API by Microsoft** that provides functions to render two-dimensional (2D) and three-dimensional (3D) graphics, and uses hardware acceleration if available on the graphics card. It was designed by Microsoft Corporation for use on the **Windows platform**.

**OpenGL is an open standard API**  that provides many functions to render 2D and 3D graphics, and is available on most modern operating systems including but not limited to **Windows, macOS, and Linux**.

Note that many essential OpenGL extensions and methods, although documented, are also patented, thus imposing serious legal troubles to implement them (see issues with Mesa[9]).

**OpenGL and Direct3D are both implemented in the display device driver**. A significant difference however is that Direct3D implements the API in a common runtime (supplied by Microsoft), which in turn talks to a low-level device driver interface (DDI). With OpenGL, every vendor implements the full API in the driver. This means that some API functions may have slightly different behavior from one vendor to the next. The GLSL shader compilers of different vendors also show slightly different behavior. The following compares the two APIs, structured around various considerations mostly relevant to game development.

OpenGL hardware acceleration on Windows is achieved by users first installing installable client drivers (ICDs) developed by GPU makers.[10] These ICDs are, in virtually all cases, bundled with the standard driver download package from the hardware vendor (IHV), so installing recent graphics drivers is sufficient to provide hardware OpenGL support

*Πηγή:https://en.wikipedia.org/wiki/Comparison\_of\_OpenGL\_and\_Direct3D*

# Προϋποθέσεις για κατανόηση της OpenGL

## Τί είναι API;

**An API is a software-to-software interface, not a user interface**. With APIs, **applications talk to each other without any user knowledge or intervention**. When you buy movie tickets online and enter your credit card information, the movie ticket Web site uses an API to send your credit card information to a remote application that verifies whether your information is correct. Once payment is confirmed, the remote application sends a response back to the movie ticket Web site saying it's OK to issue the tickets.

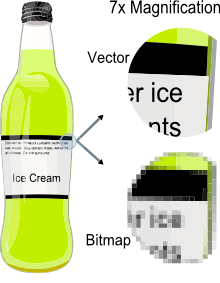
*Πηγή:* [*http://money.howstuffworks.com/business-communications/how-to-leverage-an-api-for-conferencing1.htm*](http://money.howstuffworks.com/business-communications/how-to-leverage-an-api-for-conferencing1.htm)

## 

## Τί είναι διανυσματικά γραφικά (vector graphics);

**Vector graphics** is the use of [**geometrical primitives**](https://en.wikipedia.org/wiki/Geometric_primitive) **such as** [**points**](https://en.wikipedia.org/wiki/Point_(spatial))**,** [**lines**](https://en.wikipedia.org/wiki/Line_(mathematics))**,** [**curves**](https://en.wikipedia.org/wiki/Curve), and [shapes](https://en.wikipedia.org/wiki/Shape) or [polygons](https://en.wikipedia.org/wiki/Polygons)—all of which are **based on mathematical expressions**—to **represent** [**images**](https://en.wikipedia.org/wiki/Image) **in** [**computer graphics**](https://en.wikipedia.org/wiki/Computer_graphics). Vector graphics are based on vectors (also called paths), which lead through locations called control points or nodes. Each of these points has a definite position on the x and y axes of the work plane and determines the direction of the path; further, each path may be assigned a stroke color, shape, thickness, and fill. These properties don't increase the size of vector graphics files in a substantial manner, as all information resides in the document's structure, which describes solely how the vector should be drawn. Vector graphics can be magnified infinitely without loss of quality, while pixel-based graphics cannot.

The term vector graphics is typically used only for 2D (planar) graphics objects, in order to distinguish them from 2D [raster graphics](https://en.wikipedia.org/wiki/Raster_graphics), which are also very common. [3D graphics](https://en.wikipedia.org/wiki/3D_graphics) as commonly implemented today (e.g., in [OpenGL](https://en.wikipedia.org/wiki/OpenGL)) are typically described using primitives like 3D points and polygons connecting these (which in turn describe surfaces); these 3D primitives are much more similar to vector graphics than to raster graphics, but aren't explicitly called vector graphics.



*Πηγή:* [*https://en.wikipedia.org/wiki/Vector\_graphics*](https://en.wikipedia.org/wiki/Vector_graphics)

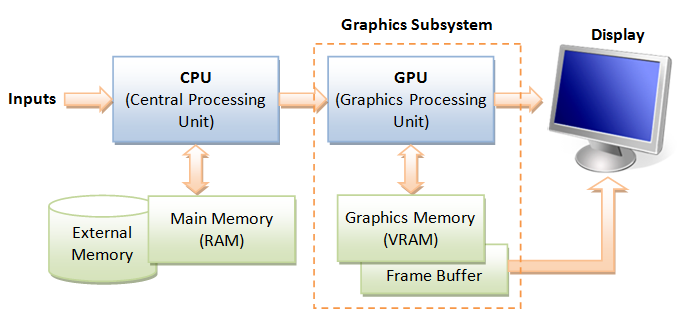
## 

## 

## 

## 

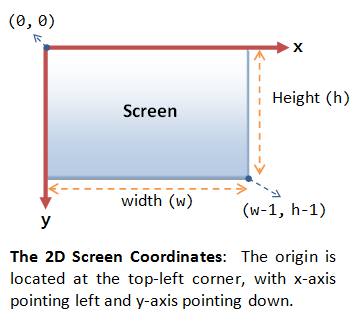
## Πώς δουλεύει το Computer Graphics Hardware

****

**GPU (Graphics Processing Unit)**

Modern day computer has dedicated Graphics Processing Unit (GPU) to produce images for the display, with its own graphics memory (or Video RAM or VRAM).

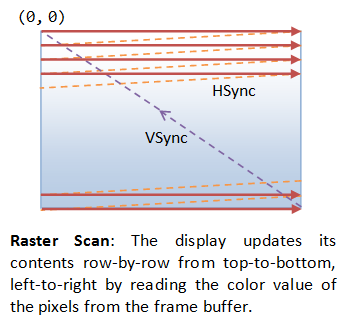
**Pixels and Frame**

****

**All modern displays are raster-based**. A **raster is a 2D rectangular grid of pixels** (or picture elements). A pixel has two properties: a color and a position. Color is expressed in RGB (Red-Green-Blue) components - typically 8 bits per component or 24 bits per pixel (or *true color*). The position is expressed in terms of (x, y) coordinates. The origin (0, 0) is located at the top-left corner, with x-axis pointing right and y-axis pointing down. This is different from the conventional 2D Cartesian coordinates, where y-axis is pointing upwards.

The number of color-bits per pixel is called the ***depth*** (or *precision*) of the display. The number of rows by columns of the rectangular grid is called the *resolution* of the display, which can range from 640x480 (VGA), 800x600 (SVGA), 1024x768 (XGA) to 1920x1080 (FHD), or even higher.

**Frame Buffer and Refresh Rate**

****

The color values of the pixels are stored in a special part of graphics memory called *frame buffer*. The GPU writes the color value into the frame buffer. The display reads the color values from the frame buffer row-by-row, from left-to-right, top-to-bottom, and puts each of the values onto the screen. This is known as *raster-scan*. The display refreshes its screen several dozen times per second, typically 60Hz for LCD monitors and higher for CRT tubes. This is known as the*refresh rate*.

A complete screen image is called a *frame*.

**Double Buffering and VSync**

While the display is reading from the frame buffer to display the current frame, we might be updating its contents for the next frame (not necessarily in raster-scan manner). This would result in the so-called *tearing*, in which the screen shows parts of the old frame and parts of the new frame.

This could be resolved by using so-called *double buffering*. Instead of using a single frame buffer, modern GPU uses two of them: a *front buffer* and a *back buffer*. The display reads from the front buffer, while we can write the next frame to the back buffer. When we finish, we signal to GPU to swap the front and back buffer (known as *buffer swap* or *page flip*).

Double buffering alone does not solve the entire problem, as the buffer swap might occur at an inappropriate time, for example, while the display is in the middle of displaying the old frame. This is resolved via the so-called *vertical synchronization* (or *VSync*) at the end of the raster-scan. When we signal to the GPU to do a buffer swap, the GPU will wait till the next VSync to perform the actual swap, after the entire current frame is displayed.

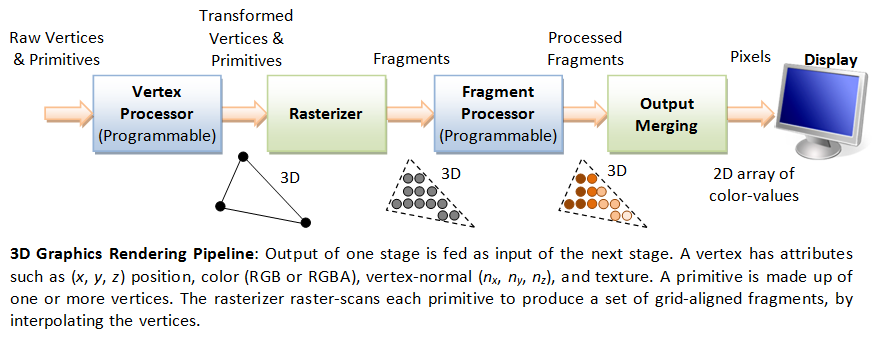
The most important point is: When the VSync buffer-swap is enabled, you cannot refresh the display faster than the refresh rate of the display!!! For the LCD/LED displays, the refresh rate is typically locked at 60Hz or 60 frames per second, or 16.7 milliseconds for each frame. Furthermore, if you application refreshes at a fixed rate, the resultant refresh rate is likely to be an integral factor of the display's refresh rate, i.e., 1/2, 1/3, 1/4, etc.

## Τι είναι 3D Graphics Rendering Pipeline;

A *pipeline*, in computing terminology, refers to a series of processing stages in which the output from one stage is fed as the input of the next stage, similar to a factory assembly line or water/oil pipe. With massive parallelism, pipeline can greatly improve the overall throughput.

In computer graphics, *rendering* is the process of producing image on the display from model description.

The *3D Graphics Rendering Pipeline* accepts description of 3D objects in terms of vertices of primitives (such as triangle, point, line and quad), and produces the color-value for the pixels on the display.



The 3D graphics rendering pipeline consists of the following main stages:

1. Vertex Processing: Process and transform individual vertices.
2. Rasterization: Convert each primitive (connected vertices) into a set of fragments. A fragment can be treated as a pixel in 3D spaces, which is aligned with the pixel grid, with attributes such as position, color, normal and texture.
3. Fragment Processing: Process individual fragments.
4. Output Merging: Combine the fragments of all primitives (in 3D space) into 2D color-pixel for the display.

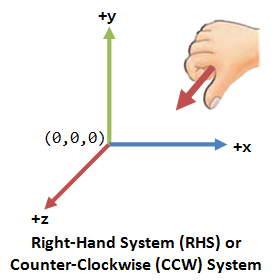
In modern GPUs, the vertex processing stage and fragment processing stage are programmable. You can write programs, known as *vertex shader* and *fragment shader* to perform your custom transform for vertices and fragments. The shader programs are written in C-like high level languages such as GLSL (OpenGL Shading Language), HLSL (High-Level Shading Language for Microsoft Direct3D), or Cg (C for Graphics by NVIDIA).

On the other hand, the rasterization and output merging stages are not programmable, but configurable - via configuration commands issued to the GPU.

# Βασικές συνιστώσες της OpenGL

## Σύστημα συντεταγμένων, Vertices, Primitives, Fragment and Pixels

**3D Graphics Coordinate Systems**

****

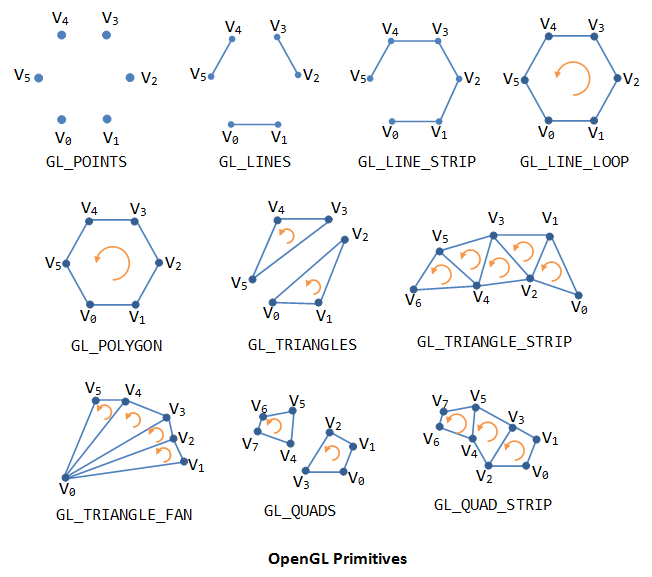
OpenGL adopts the *Right-Hand Coordinate System* (RHS). In the RHS, the x-axis is pointing right, y-axis is pointing up, and z-axis is pointing out of the screen. With your right-hand fingers curving from the x-axis towards the y-axis, the thumb is pointing at the z-axis. RHS is *counter-clockwise* (CCW). The 3D Cartesian Coordinates is a RHS.

Some graphics software (such as Microsoft Direct3D) use *Left-hand System* (LHS), where the z-axis is inverted. LHS is clockwise (CW). In this article, we shall adopt the RHS and CCW used in OpenGL.

**Primitives**

The inputs to the Graphics Rendering Pipeline are geometric *primitives* (such as triangle, point, line or quad), which is formed by *one or more vertices*.

OpenGL supports three classes of geometric primitives: *points*, *line segments*, and *closed polygons*. They are specified via vertices. Each vertex is associated with its attributes such as the position, color, normal and texture. OpenGL provides 10 primitives as shown. Sphere, 3D box and pyramid are not primitives. They are typically assembled using primitive triangle or quad.



OpenGL interprets the vertices and renders each primitive using the following rules:

GL\_POINTS— Use this primitive type to render mathematical points. OpenGL renders a point for each vertex specified.

GL\_LINES— Use this primitive to draw unconnected line segments. OpenGL draws a line segment for each group of two vertices. If the application specifies n vertices, OpenGL renders n/2 line segments. If n is odd, OpenGL ignores the final vertex.

GL\_LINE\_STRIP— Use this primitive to draw a sequence of connected line segments. OpenGL renders a line segment between the first and second vertices, between the second and third, between the third and fourth, and so on. If the application specifies n vertices, OpenGL renders n–1 line segments.

GL\_LINE\_LOOP— Use this primitive to close a line strip. OpenGL renders this primitive like a GL\_LINE\_STRIP with the addition of a closing line segment between the final and first vertices.

GL\_TRIANGLES— Use this primitive to draw individual triangles. OpenGL renders a triangle for each group of three vertices. If your application specifies n vertices, OpenGL renders n/3 triangles. If n isn't a multiple of 3, OpenGL ignores the excess vertices.

GL\_TRIANGLE\_STRIP— Use this primitive to draw a sequence of triangles that share edges. OpenGL renders a triangle using the first, second, and third vertices, and then another using the second, third, and fourth vertices, and so on. If the application specifies n vertices, OpenGL renders n–2 connected triangles. If n is less than 3, OpenGL renders nothing.

GL\_TRIANGLE\_FAN— Use this primitive to draw a fan of triangles that share edges and also share a vertex. Each triangle shares the first vertex specified. If the application specifies a sequence of vertices v, OpenGL renders a triangle using v 0, v 1, and v 2; another triangle using v 0, v 2, and v 3; another triangle using v 0, v 3, and v 4; and so on. If the application specifies n vertices, OpenGL renders n–2 connected triangles. If n is less than 3, OpenGL renders nothing.

GL\_QUADS— Use this primitive to draw individual convex quadrilaterals. OpenGL renders a quadrilateral for each group of four vertices. If the application specifies n vertices, OpenGL renders n/4 quadrilaterals. If n isn't a multiple of 4, OpenGL ignores the excess vertices.

GL\_QUAD\_STRIP— Use this primitive to draw a sequence of quadrilaterals that share edges. If the application specifies a sequence of vertices v, OpenGL renders a quadrilateral using v 0, v 1, v 3, and v 2; another quadrilateral using v 2, v 3, v 5, and v 4; and so on. If the application specifies n vertices, OpenGL renders (n-2)/2 quadrilaterals. If n is less than 4, OpenGL renders nothing.

GL\_POLYGON— Use GL\_POLYGON to draw a single filled convex n-gon primitive. OpenGL renders an n-sided polygon, where n is the number of vertices specified by the application. If n is less than 3, OpenGL renders nothing.

*Πηγή:* [*http://www.informit.com/articles/article.aspx?p=461848*](http://www.informit.com/articles/article.aspx?p=461848)

**Vertices**

Recall that a primitive is made up of one or more vertices. A vertex, in computer graphics, has these attributes:

Position in 3D space V=(x, y, z): typically expressed in floating point numbers.

Color: expressed in RGB (Red-Green-Blue) or RGBA (Red-Green-Blue-Alpha) components. The component values are typically normalized to the range of 0.0 and 1.0 (or 8-bit unsigned integer between 0 and 255). Alpha is used to specify the transparency, with alpha of 0 for totally transparent and alpha of 1 for opaque.

Vertex-Normal N=(nx, ny, nz): We are familiar with the concept of surface normal, where the normal vector is perpendicular to the surface. In computer graphics, however, we need to attach a normal vector to each vertex, known as vertex-normal. Normals are used to differentiate the front- and back-face, and for other processing such as lighting. Right-hand rule (or counter-clockwise) is used in OpenGL. The normal is pointing outwards, indicating the outer surface (or front-face).

Texture T=(s, t): In computer graphics, we often wrap a 2D image to an object to make it seen realistic. A vertex could have a 2D texture coordinates (s, t), which provides a reference point to a 2D texture image.

Others.

*Το πλήρες άρθρο υπάρχει στη διεύθυνση:* [*https://www.ntu.edu.sg/home/ehchua/programming/opengl/CG\_BasicsTheory.html*](https://www.ntu.edu.sg/home/ehchua/programming/opengl/CG_BasicsTheory.html)

# Βιβλιοθήκες OpenGL

OpenGL (Open Graphics Library) is a cross-platform, hardware-accelerated, language-independent, industrialstandard API for producing 3D (including 2D) graphics. Modern computers have dedicated GPU (GraphicsProcessing Unit) with its own memory to speed up graphics rendering. OpenGL is the software interface tographics hardware. In other words, OpenGL graphic rendering commands issued by your applications could bedirected to the graphic hardware and accelerated.We use 3 sets of libraries in our OpenGL programs:

1. Core OpenGL (GL): consists of hundreds of commands, which begin with a prex "gl" (e.g. glColor, glVertex, glTranslate, glRotate). The Core OpenGL models an object via a set of geometric primitives such as point, line and polygon.
2. OpenGL Utility Library (GLU): built on-top of the core OpenGL to provide important utilities (such assetting camera view and projection) and more building models (such as qradric surfaces and polygontessellation). GLU commands start with a prex "glu" (e.g., gluLookAt, gluPerspective).
3. OpenGL Utilities Toolkit (GLUT): OpenGL is designed to be independent of the windowing system oroperating system. GLUT is needed to interact with the Operating System (such as creating a window,handling key and mouse inputs); it also provides more building models (such as sphere and torus). GLUTcommands start with a prex of "glut" (e.g., glutCreatewindow, glutMouseFunc). GLUT is platform independent, which is built on top of platform-specic OpenGL extension such as GLX for X Window System,WGL for Microsoft Window, and AGL, CGL or Cocoa for Mac OS.Quoting from theopengl.org: "GLUT is designed for constructing small to medium sized OpenGL programs.While GLUT is well-suited to learning OpenGL and developing simple OpenGL applications, GLUT is not a full-featured toolkit so large applications requiring sophisticated user interfaces are better o using nativewindow system toolkits.

GLUT is simple, easy, and small.

"Alternative of GLUT includes SDL, ....4.

OpenGL Extension Wrangler Library (GLEW)

: "GLEW is a cross-platform open-source C/C++extension loading library. GLEW provides ecient run-time mechanisms for determining which OpenGLextensions are supported on the target platform." Source and pre-build binary available athttp://glew.sourceforge.net/. A standalone utility called "glewinfo.exe" (under the "bin" directory) can beused to produce the list of OpenGL functions supported by your graphics system.5.Others.

http://www3.ntu.edu.sg/home/ehchua/programming/opengl/CG\_Introduction.html

https://www.scribd.com/document/200570296/An-Introduction-on-OpenGL-With-2D-Graphics-OpenGL-Tutorial

## Συγγραφή κώδικα (Development) για OpenGL API

OpenGL is an evolving API. New versions of the OpenGL specifications are regularly released by the [Khronos Group](https://en.wikipedia.org/wiki/Khronos_Group), each of which extends the API to support various new features. The details of each version are decided by consensus between the Group's members, including graphics card manufacturers, operating system designers, and general technology companies such as [Mozilla](https://en.wikipedia.org/wiki/Mozilla) and [Google](https://en.wikipedia.org/wiki/Google).[[7]](https://en.wikipedia.org/wiki/OpenGL#cite_note-7)

In addition to the features required by the core API, [GPU](https://en.wikipedia.org/wiki/Graphics_processing_unit) vendors may provide additional functionality in the form of extensions. Extensions may introduce new functions and new constants, and may relax or remove restrictions on existing OpenGL functions. Vendors can use extensions to expose custom APIs without needing support from other vendors or the Khronos Group as a whole, which greatly increases the flexibility of OpenGL. All extensions are collected in, and defined by, the OpenGL Registry.[[8]](https://en.wikipedia.org/wiki/OpenGL#cite_note-OpenGLRegistryRoot-8)

Each extension is associated with a short identifier, based on the name of the company which developed it. For example, [Nvidia](https://en.wikipedia.org/wiki/Nvidia)'s identifier is NV, which is part of the extension nameGL\_NV\_half\_float, the constant GL\_HALF\_FLOAT\_NV, and the function glVertex2hNV().[[9]](https://en.wikipedia.org/wiki/OpenGL#cite_note-9) If multiple vendors agree to implement the same functionality using the same API, a shared extension may be released, using the identifier EXT. In such cases, it could also happen that the Khronos Group's Architecture Review Board gives the extension their explicit approval, in which case the identifier ARB is used.[[10]](https://en.wikipedia.org/wiki/OpenGL#cite_note-10)

## Context and window toolkits

Given that creating an OpenGL context is quite a complex process, and given that it varies between [operating systems](https://en.wikipedia.org/wiki/Operating_system), automatic OpenGL context creation has become a common feature of several game-development and user-interface [libraries](https://en.wikipedia.org/wiki/Library_(computing)), including [SDL](https://en.wikipedia.org/wiki/Simple_DirectMedia_Layer), [Allegro](https://en.wikipedia.org/wiki/Allegro_(software)), [SFML](https://en.wikipedia.org/wiki/SFML), [FLTK](https://en.wikipedia.org/wiki/FLTK), and [Qt](https://en.wikipedia.org/wiki/Qt_(framework)). A few libraries have been designed solely to produce an OpenGL-capable window. The first such library was [GLUT](https://en.wikipedia.org/wiki/OpenGL_Utility_Toolkit) (later superseded by [freeglut](https://en.wikipedia.org/wiki/Freeglut)). [GLFW](https://en.wikipedia.org/wiki/GLFW) is a newer alternative.[[11]](https://en.wikipedia.org/wiki/OpenGL#cite_note-11)

These toolkits are designed specifically around creating and managing OpenGL windows. They also manage input, but little beyond that.[[12]](https://en.wikipedia.org/wiki/OpenGL#cite_note-12)

* [GLFW](https://en.wikipedia.org/wiki/GLFW) — A crossplatform windowing and keyboard/mouse/joystick handler. Is more aimed for creating games.
* [freeglut](https://en.wikipedia.org/wiki/Freeglut) — A crossplatform windowing and keyboard/mouse handler. Its API is a superset of the GLUT API, and it is more stable and up to date than GLUT.
* [GLUT](https://en.wikipedia.org/wiki/OpenGL_Utility_Toolkit) — An old windowing handler, no longer maintained.

Several "multimedia libraries" can create OpenGL windows, in addition to input, sound and other tasks useful for game-like applications.

* [Allegro 5](https://en.wikipedia.org/wiki/Allegro_(software)) — A cross-platform multimedia library with a C API focused on game development.
* [SDL](https://en.wikipedia.org/wiki/Simple_DirectMedia_Layer) — A cross-platform multimedia library with a C API.
* [SFML](https://en.wikipedia.org/wiki/SFML) — A cross-platform multimedia library with a C++ API.

*Πηγή:* [*https://en.wikipedia.org/wiki/OpenGL#Development*](https://en.wikipedia.org/wiki/OpenGL#Development)

# GLUT

The OpenGL Utility Toolkit (GLUT) is a [library](https://en.wikipedia.org/wiki/Library_(computing)) of utilities for [OpenGL](https://en.wikipedia.org/wiki/OpenGL) programs, which primarily perform system-level [I/O](https://en.wikipedia.org/wiki/Input/output) with the host[operating system](https://en.wikipedia.org/wiki/Operating_system). Functions performed include window definition, window control, and monitoring of [keyboard](https://en.wikipedia.org/wiki/Computer_keyboard) and [mouse](https://en.wikipedia.org/wiki/Computer_mouse) input. Routines for drawing a number of geometric primitives (both in solid and wireframe mode) are also provided, including [cubes](https://en.wikipedia.org/wiki/Cube_(geometry)), [spheres](https://en.wikipedia.org/wiki/Sphere) and the [Utah teapot](https://en.wikipedia.org/wiki/Utah_teapot). GLUT also has some limited support for creating pop-up menus.

GLUT was written by [Mark J. Kilgard](https://en.wikipedia.org/wiki/Mark_Kilgard), author of OpenGL Programming for the X Window System and The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics, while he was working for [Silicon Graphics](https://en.wikipedia.org/wiki/Silicon_Graphics) Inc.

The two aims of GLUT are to allow the creation of rather portable code between operating systems (GLUT is [cross-platform](https://en.wikipedia.org/wiki/Cross-platform)) and to make learning OpenGL easier. Getting started with OpenGL programming while using GLUT often takes only a few lines of code and does not require knowledge of operating system–specific windowing [APIs](https://en.wikipedia.org/wiki/Application_programming_interface).

*Πηγή:* [*https://en.wikipedia.org/wiki/OpenGL\_Utility\_Toolkit*](https://en.wikipedia.org/wiki/OpenGL_Utility_Toolkit)