Allegro

Мартин Красимиров Кирилов 11 А
'' Технологично училище “Електронни системи” към ТУ-София''
www.elsys-bg.org

Allegro 4 and Allegro 5 are cross-platform, libraries mainly aimed at video game and multimedia programming. They handle common, low-level tasks such as creating windows, accepting user input, loading data, drawing images, playing sounds, etc. and generally abstracting away the underlying platform. However, Allegro is *not* a game engine: you are free to design and structure your program as you like.

According to the Oxford Companion to Music, Allegro is the Italian for «quick, lively, bright». *It is also a recursive acronym which stands for «Allegro Low LEvel Game ROutines». Allegro was started by Shawn Hargreaves in the mid-90's but has since received* contributions from hundreds of people over the net.

Allegro 5

Allegro 5 is the latest major revision of the library, designed to take advantage of modern hardware (e.g. hardware acceleration using 3D cards) and operating systems. Although it is not backwards compatible with earlier versions, it still occupies the same niche and retains a familiar style.

Allegro 5.0 supports the following platforms:

Unix/Linux

Windows (MSVC, MinGW)

MacOS X

iPhone

Allegro 5.1 also adds support for:

Android

Allegro only supports 2D graphics primitives natively, but it is perfectly reasonable to use Allegro alongside a 3D API (e.g. OpenGL, Direct3D, and higher level libraries), while Allegro handles the other tasks. Allegro is also designed to be modular; e.g. if you prefer, you can substitute another audio library.

Allegro 4

Allegro 4 is the classic library, whose API is backwards compatible all the way back to Allegro 2.0 for DOS/DJGPP (1996). It is no longer *actively* developed, but we still apply patches sent to us by contributors, mainly to fix minor bugs. Every so often we will make new releases.

Allegro 4.4 supports the following platforms:

Unix/Linux

Windows (MSVC, MinGW, Cygwin)

MacOS X

Haiku/BeOS

PSP (currently in SVN repository only)

The older Allegro 4.2 branch additionally supports:

Windows (Borland)

QNX

DOS (DJGPP, Watcom)

Language bindings

Although Allegro is written in C and C++, and is mainly designed to be used from such, it's possible to use Allegro from other programming languages.

Bindings for Allegro 5

allua

Lua binding to Allegro 5 by Trezker.

Chicken Scheme

Binding for Chicken Scheme by Daniel Leslie.

DAllegro5

D binding to Allegro 5 by SiegeLord.

Python

The Allegro 5 distribution comes with a basic Python wrapper; see the python directory.

Bindings for Allegro 4

C++

alBitmap

alBitmap is a C++ Wrapper for the Allegro BITMAP datatype. Features built-in memory allocation/deallocation, inlined functions for bitmap creation, loading, saving, blitting, comparing, plus some convenience functions for printing text with STL's string type.

Allegro Simplificator

Allegro Simplificator is a C++ wrapper. It provides classes for most Allegro functions and adds some new functionality (cross-platform networking). It comes with some examples and extensive documentation.

C

AllegNet

AllegNet is a .NET library, coded in C# 2.0 based on Allegro 4.2.0. With AllegNet you can build your own video game in full managed code. So you can use this to build C#, vb.net, J# and C++ managed games.

Mallegro

Mallegro (or Managed Allegro) is a .NET wrapper for Allegro written by Michael Jensen. It promises making Allegro available to C# and VB.NET developers.

D

DAllegro

Tydr Schnubbis organised an effort to be able to use Allegro from the D programming language.

Lisp

Common Lisp FFI for Allegro

cl-alleg provides a foreign function interface for the Allegro games library. This is built on top of CFFI and so should be portable across multiple common lisp implementations. The author also writes a cl-alleg game development blog at http://gameylisp.blogspot.com/.

Lua

LuAllegro

The goal of the LuAllegro project is to provide a free multi-platform graphics library for Lua. Of course, Allegro is used for this purpose.

Pascal

Allegro.pas

Allegro.pas is a wrapper to use Allegro with Pascal compilers like Delphi or Free Pascal.

Perl

AlPerl

A Perl interface to Allegro written by Colin O'Leary.

Python

PyAllegro

This page hosts all available Python bindings. There are currently two projects: Alpy (a pure C interface) and PyAlleg (a Pyrex interface).

Allegro Basics

The best way to start using Allegro is to download and install the DEV C++ IDE for Windows. After this you can install Allegro libraries and start using it. First of all open a new project in Dev C++ and pick “Windows Application”.Then click on “Project”, “Project Options”, “Paremeters”, “Add library or Object” and set this to “C:\Dev-Cpp\lib\liballeg.a”.This will let the compiler know where to find the Allegro functions.

Here is a simple Allegro program:

```
#include <allegro.h>

int main(){

allegro_init();
install_keyboard();
set_gfx_mode( GFX_AUTODETECT,
640,
480, 0, 0);

readkey();

return 0;
}
END_OF_MAIN();
```

This program will change the screen resolution to 640x480 and then wait for the user to press a key and then exit.The main difference from the basic C++ is that you don’t need to include iostream, because of allegro’s integrated functions. In order to startup Allegro, you have to run the allegro_init() function. install_keyboard() prepares Allegro to take user input from the keyboard.

set_gfx_mode() changes the screen resolution, it takes five parameters. The first should always be set to "GFX_AUTODETECT". The next two will be the size of the viewable screen. The last two are used for programs that require even more space.

And now after you know how to initialize allegro here is some example code of a shape drawing program:

```
#include <allegro.h>

int main(){

 allegro_init();
install_keyboard();
set_gfx_mode( GFX_AUTODETECT,
640,
480, 0, 0);

acquire_screen();

clear_to_color( screen, makecol(
255, 255, 255));

putpixel( screen, 5, 5, makecol(
128, 200, 23));

circle( screen, 20, 20, 10,
makecol( 255, 0, 0));
circlefill( screen, 50, 50, 25,
makecol( 0,255, 0));

rect( screen, 70, 70, 90, 90,
makecol( 0, 0, 255));
rectfill( screen, 100, 100, 120,
120, makecol( 12, 34, 200));

line( screen, 130, 130, 150,
150, makecol( 255, 0, 0));
line( screen, 130, 130, 170,
130, makecol( 255, 0, 0));
line( screen, 170, 130, 150,
150, makecol( 255, 0, 0));

floodfill( screen, 140, 135,
makecol( 255, 255, 0));

triangle( screen, 200, 200, 200,
220, 220, 210, makecol( 213, 79,
40));

release_screen();

readkey();

return 0;

}
END_OF_MAIN();
```

This program will change the background color to white and draw several shapes on the screen.

"clear_to_color(screen, makecol(255, 255, 255));" is used to clear any bitmap specified in the first parameter to the color in the second.

All the functions used to draw here take the bitmap to draw to as the first parameter. putpixel() takes four parameters, the second and third define the x and y coordinates. The fourth the color of the pixel. A pixel is the smallest component in an image, what you see on your monitor is a bunch of different color pixels, the size of the pixel depends on the resolution you set with set_gfx_mode().

circle() and circlefill() both draw circles, the difference is that circlefill() will fill in the circle with the color specified in the last parameter while circle() simply draws the outline. The second and third parameters specify the position. The fourth defins the radius of the circle.

rect() and rectfill() draw rectangles which start at the point defined by the second and third parameters and end at the point defined by the fourth and fith parameters. The difference between rect() and rectfill() is the same as the two circle functions.

line() draws a line of the color specified in the last parameter starting at the point defined by the second and third parameter and ending at the point defined by the fourth and fith.

floodfill() works like the "Paint Bucket" tool found in most image editing software. It will fill an area starting at the point defined by the second and third pixel with the color specified in the fourth.

triangle() draws a colored in triangle, there is no trianglefill(). The three points of the triangle are defined by second through sixth parameters. The color is defined by the last parameter.