Chuẩn bị môi trường lập trình OpenGL

Mục lục

Giới thiệu

OpenGL là một thư viện lập trình đồ họa thuần tuý. Thư viện lập trình đồ họa này có thể vẽ bất kỳ thứ gì mà ta yêu cầu. Tuy nhiên, OpenGL lại không có các tập lệnh để xử lý việc xuất/nhập từ bàn phím, chuột, tập tin hoặc thao tác trên các cửa sổ ứng dụng, quản lý các tập mở rộng OpenGL, đọc/ghi các tập tin hình ảnh... May mắn là hiện nay, có khá nhiều các thư viện hỗ trợ kèm theo như *FreeGLUT*, *SDL*, *SDL2*, *SFML*, *GLFW*, *GLEW*, *SOIL*... Các thư viện lập trình này ngoài tính năng là các thư viện đồ họa, chúng còn cho phép ta xử lý dễ dàng các vấn đề nêu trên. Hơn nữa, các thư viện lập trình bổ sung này không phụ thuộc nền tảng hệ điều hành, có nghĩa là ngoài Windows, chúng còn có thể hoạt động trên cả Linux và OSX. Đây là điều mà các lập trình viên OpenGL mong muốn vì bản chất của OpenGL là độc lập nền tảng.

Trong loạt bài tập này, ta lựa chọn sử dụng thư viện SDL2 và GLEW để phục vụ cho các ứng dụng OpenGL sẽ được xây dựng. Ngôn ngữ lập trình được lựa chọn sử dụng xuyên suốt loạt bài tập thực hành này là `C++`.

Mục tiêu

Một số hướng dẫn để thiết lập các thông số cho trình biên dịch, cài đặt các thư viện hỗ trợ lập trình SDL2 và GLEW phục vụ cho việc xây dựng các ứng dụng OpenGL trên các trình biên dịch như Visual Studio 2015, MinGW trên nền Windows, GNU C/C++ trên nền Linux và OSX. Trong các phần được trình bày dưới đây, ta chỉ cần

tìm hiểu và thực hiện phần nội dung có liên quan đến trình biên dịch và nền tảng hệ điều hành mà ta đang sử dụng để triển khai các ứng dụng OpenGL mà thôi, không nhất thiết phải thực hiện toàn bộ các hướng dẫn này trừ khi ta cần triển khai các ứng dụng OpenGL trên tất cả các nền tảng đó.

Thư viện SDL2

Ta truy cập vào trang chủ của thư viện SDL2 để tải về gói thư viện phát triển (*development libraries*) và tiến hành cài đặt thư viện này vào hệ thống. Địa chỉ của trang chủ thư viện SDL2 là `https://www.libsdl.org/`.

Lưu ý: ta chỉ tải về bộ *development library* ứng với nền tảng hệ điều hành và trình biên dịch mà ta đang sử dụng mà thôi. Phiên bản tại thời điểm biên soạn bài tập này là SDL 2.0.8.

Thư viện GLEW

Tương tự SDL2, để sử dụng thư viện lập trình GLEW, ta truy cập vào liên kết `http://glew.sourceforge.net` để tải về bộ thư viện lập trình này và cài đặt vào hệ thống. Phiên bản hiện tại của GLEW là 2.1.0.

Lưu ý, nếu hệ điều hành OSX là hệ điều hành được sử dụng để viết các ứng dụng OpenGL, ta có thể bỏ qua quá trình cài đặt thư viện này vì *framework* OpenGL của hệ điều hành đã đảm nhiệm vai trò của GLEW.

Môi trường lập trình Visual Studio 2015 (14.0)

Cài đặt SDL2 và GLEW

Tải về tập tin `SDL2-devel-2.0.8-VC.zip` và `glew-2.1.0-vc-win32.zip`, giải nén chúng và các thư mục riêng biệt và thực hiện các thao tác sau:

a. Đối với SDL2:

- Sao chép các tập tin `.h` từ thư mục `include` của gói SDL2 đã giải nén vào trong thư mục `C:\Program Files (x86)\Microsoft Visual Studio 14.0\VC\include\SDL2`.
- Sao chép các tập tin `.lib` cho kiến trúc `x86` trong thư mục `lib\x86` vào thư mục `C:\Program Files (x86)\Microsoft Visual Studio 14.0\VC\lib`.
- Tương tự, sao chép các tập tin `.lib` cho kiến trúc `64-bit` bên trong thư mục `lib\x64` vào thư mục `C:\Program Files (x86)\Microsoft Visual Studio 14.0\VC\lib\amd64`.
- Đối với các thư viện liên kết động `.dll`, ta copy tập tin `SDL2.dll` cho kiến trúc `x86` vào thư mục `C:\Windows\SysWOW64` và tập tin `SDL2.dll` cho kiến trúc `64-bit` vào thư mục `C:\Windows\System32`.

b. Đối với GLEW:

- Sao chép tập tin `glew32.dll` trong thư mục `bin\Release\Win32` đã giải nén vào thư mục `C:\Windows\SysWOW64`.

- Sao chép tập tin `glew32.dl` trong thư mục `bin\Release\x64` vào thư mục `C:\Windows\System32`.
- Sao chép toàn bộ thư mục `GL` trong thư mục `include` vào thư mục `C:\Program Files (x86)\Microsoft Visual Studio 14.0\VC\include\`.
- Sao chép các tập tin `.lib` (`glew32.lib`, `glew32s.lib`) từ thư mục `lib\Release\Win32` vào thư mục `C:\Program Files (x86)\Microsoft Visual Studio 14.0\VC\lib\`.
- Sao chép các tập tin `.lib` (`glew32.lib`, `glew32s.lib`) từ thư mục `lib\Release\x64` vào thư mục `C:\Program Files (x86)\Microsoft Visual Studio 14.0\VC\lib\amd64\`.

Biên dịch các project có sử dụng SDL2 và GLEW

Để tiện trong việc biên dịch các ứng dụng OpenGL có sử dụng SDL2 và GLEW từ Visual Studio, ta có thể sử dụng các chỉ thị liên kết trực tiếp trong mã nguồn như sau mà không cần phải thiết lập thông qua chức năng *Properties* của project phần mềm:

```
#ifdef _MSC_VER
#      pragma comment (lib, "SDL2.lib")          // thư viện SDL2
#      pragma comment (lib, "SDL2Main.lib")      // thư viện chứa hàm SDL2_Main
#      pragma comment (lib, "glew32.lib")        // thư viện GLEW
#      pragma comment (lib, "opengl32.lib")      // thư viện OpenGL
#endif
```

Đoạn mã trên thường được đặt ngay sau phần chỉ thị `#include` danh sách các thư viện. Và do chỉ có Visual Studio mới hỗ trợ liên kết thư viện theo cách này nên ta cần đặt chúng bên trong cặp chỉ thị `#ifdef _MSC_VER` và `#endif`. Điều này giúp cho các trình biên dịch khác bỏ qua các chỉ thị liên kết trực tiếp này mà không thông báo lỗi. Khi đó, ta chỉ cần biên dịch project ứng dụng OpenGL một cách dễ dàng như biên dịch các project phần mềm khác.

Môi trường lập trình MinGW32

Cài đặt SDL2 và GLEW

Ta tải về tập tin `SDL2-devel-2.0.8-mingw.tar.gz` và giải nén vào một thư mục riêng biệt. Đối với GLEW, thư viện lập trình này không cung cấp gói thư viện đã được biên dịch sẵn cho MinGW32, vì vậy ta hoặc liên hệ với giảng viên phụ trách môn học để được sao chép gói thư viện này hoặc có thể tìm hiểu cách thức biên dịch để tự mình có thể biên dịch hoàn chỉnh thư viện GLEW trước khi sử dụng với MinGW32. Phần nội dung tiếp theo sau đây giả sử ta đã biên dịch thành công GLEW và tiến hành cài đặt thư viện này vào hệ thống. Ta cũng giả sử MinGW32 được cài đặt tại `C:\MinGW`.

a. Đối với SDL2

- Copy các tập tin `.h` trong thư mục `i686-w64-mingw32/include/SDL2` vào thư mục `C:\MinGW\include\SDL2`.
- Copy các tập tin `.a` và `.la` trong thư mục `i686-w64-mingw32/lib` vào thư mục `C:\MinGW\lib`.

- Copy tập tin `SDL2.dll` trong thư mục `i686-w64-mingw32/bin` vào thư mục `C:\Windows\SysWOW64`.

b. Đối với GLEW

- Sao chép tập tin `glew32.dll` vào thư mục `C:\Windows\System32` hoặc `C:\Windows\SysWOW64` tùy thuộc vào kiến trúc mã lệnh đã được sử dụng để phát sinh thư viện `glew32.dll`.
- Sao chép toàn bộ thư mục `GL` bên trong thư mục include vào `C:\MinGW\include`.
- Sao chép toàn bộ các tập tin `.a` (`libglew32.a`, `libglew32.dll.a`) vào thư mục `C:\MinGW\lib`.

Biên dịch các project có sử dụng SDL2 và GLEW

Mặc định MinGW không cung cấp một môi trường phát triển tích hợp như Visual Studio, vì vậy để biên dịch các ứng dụng OpenGL với SDL2 và GLEW, ta có hai lựa chọn:

- Nhập trực tiếp các lệnh biên dịch từ cửa sổ dòng lệnh `CMD`. Cú pháp biên dịch như sau:

```
g++ -std=c++11 lab01.cpp -o lab01.exe -lmingw32 -lSDL2Main -lSDL2 -lopengl32 -lglew32
```

  Trong đó:

  - Tập tin `lab01.cpp` là tập tin chứa mã nguồn cần biên dịch.
  - Tập tin `lab01.exe` là tập tin mã máy sau khi biên dịch thành công.
  - `-lmingw32`, `-lSDL2Main`, `-lSDL2`, `-lopengl32`, `-lglew32`: liên kết các thư viện lập trình vào ứng dụng OpenGL sau khi biên dịch.
- Sử dụng `MAKEFILE` và tiện ích `MinGW32-make`: tập tin `MAKEFILE` có nội dung tổng quát như sau đây, để biên dịch ứng dụng, ta chỉ cần nhập dòng lệnh `mingw32-make` tại cửa sổ dòng lệnh `CMD`. Để sử dụng `MAKEFILE` này cho một bài tập khác, ta thay tên tập tin mã nguồn của bài tập đó vào giá trị tại macro `SOURCE=`.

```
CXX=g++                     # trình biên dịch C++
CFLAGS=-Wall -std=c++11     # sử dụng chuẩn C++11 và hiển thị tất cả các cảnh báo
LFLAGS=-lmingw32 -lSDL2Main -lSDL2 -lopengl32 -glu32 -lglew32
SOURCE=lab01                # ta thay tên tập tin mã nguồn ở đây.

all: $(SOURCE).cpp          # phụ thuộc vào tập tin lab01.cpp
    $(CXX) $(CFLAGS) $< -o $(SOURCE) $(LFLAGS)

clean:                      # gõ 'mingw32-make clean' để thực hiện đoạn lệnh này
    del $(SOURCE).exe
```

Trình biên dịch GNU C/C++ trên Ubuntu Linux

Cài đặt SDL2 và GLEW

a. Đối với SDL2

Trên môi trường Linux, cụ thể là *Ubuntu*, quá trình cài đặt thư viện SDL2 khá đơn giản và chỉ cần thực hiện đúng một câu lệnh: `sudo apt-get install libsdl2-dev`. Kế đến, ta nhập mật khẩu tài khoản quản trị để được phép cài đặt thư viện này vào hệ điều hành. Lưu ý ta cần có kết nối Internet để trình quản lý gói `apt` của Ubuntu tải về các gói thư viện cần thiết.

b. Đối với GLEW

Tương tự môi trường MinGW32, GLEW cũng không cung cấp gói thư viện biên dịch sẵn trên Ubuntu Linux, vì vậy ta cần biên dịch thư viện này từ mã nguồn. Chi tiết cách thức biên dịch GLEW trên Linux có thể được tham khảo tại địa chỉ: `http://glew.sourceforge.net/build.html`.

Biên dịch các project có sử dụng SDL2 và GLEW

Quá trình biên dịch các ứng dụng đồ họa OpenGL kết hợp với SDL2 và GLEW trên Ubuntu Linux cũng được thực hiện thông qua trình biên dịch GNU C/C++ trên cửa sổ Terminal và cũng có thể thực hiện thông qua 2 cách tương tự phần Biên dịch các project có sử dụng SDL2 và GLEW với MinGW32. Chỉ có một khác biệt nhỏ đó là phần liên kết thư viện OpenGL tham số `-lopengl32` trở thành `-lGL`, tham số `-lglew32` trở thành `-lglew`. Chi tiết như sau:

```
g++ -std=c++11 lab01.cpp -o lab01.exe -lSDL2Main -lSDL2 -lGL -lglew
```

Trình biên dịch GNU C/C++ trên OSX

Cài đặt SDL2

Như đã trình bày, trên nền hệ điều hành OSX, ta không cần thiết phải sử dụng đến thư viện GLEW, vì vậy phần nội dung này sẽ không được trình bày ở đây mà chỉ nêu cách thức cài đặt, sử dụng SDL2 mà thôi.

Quá trình cài đặt SDL2 trên hệ điều hành OSX khá đơn giản, ta tải về tập tin `SDL2-2.0.8.dmg`, double-click để mount tập tin này và copy tập tin `SDL2.framework` vào thư mục `/Library/Framework`. Lưu ý rằng ta cần phải nhập mật khẩu tài của khoản quản trị để thực hiện điều này.

Biên dịch các project có sử dụng SDL2

Đối với OSX, việc biên dịch các ứng dụng OpenGL sử dụng SDL2 thông qua GNU C/C++ cũng khá tương tự việc biên dịch trên môi trường MinGW và chỉ có một ít khác biệt. Trong đó, thư viện lập trình OpenGL và SDL2 trở thành các *framework*

của hệ điều hành OSX, vì vậy ta sử dụng tham số `-framework <tên thư viện>` thay cho tham số `-l<tên thư viện>`. Cú pháp biên dịch như sau:

```
g++ -std=c++11 lab01.cpp -o lab01 -framework OpenGL -framework SDL2
```

Sử dụng SDL2 và GLEW

Sau khi đã cài đặt thành công SDL2 và GLEW vào hệ thống, ta có thể sử dụng các thư viện này thông qua chỉ thị `#include` ở đầu chương trình như sau:

```
#include <GL/glew.h>
#include <SDL2/SDL.h>
```

Lưu ý:

- Ta chỉ cần include thư viện `glew.h` mà thôi, ngoài ra không cần include thêm các thư viện `gl.h` hoặc `glu.h` vì chúng đã được include tự động trong `glew.h`.
- Trong một số trường hợp, tập tin `glew.h` phải xuất hiện trước `SDL.h` trong danh sách include.

Một ví dụ minh họa

Trong phần này ta thực hiện một ứng dụng đồ họa OpenGL đơn giản kết hợp sử dụng SDL2 và GLEW. Nhiệm vụ của ứng dụng này là khởi tạo môi trường đồ họa OpenGL sử dụng SDL2, khởi tạo hệ thống tập mở rộng OpenGL thông qua GLEW và vẽ ra một tam giác trên cửa sổ dựng hình OpenGL. Quá trình kiểm soát vòng lặp ứng dụng sẽ được thực hiện thông qua SDL2.

a. Tập tin `LAB01.CPP`:

```
#ifndef __APPLE__
#   include <GL/glew.h>                 // Sử dụng GLEW nếu không phải OSX
#else                                   // ngược lại, include các header OpenGL chuẩn
#   include <OpenGL/gl.h>               // - OSX OpenGL header
#   include <OpenGL/glu.h>              // - OSX OpenGL Utility header
#endif
#include <SDL2/SDL.h>                   // header cho thư viện SDL2
#include <iostream>                     // cần cho hàm std::cout

#ifdef _MSC_VER                         // phần dành riêng cho MSVC
#   pragma comment(lib, "opengl32.lib") // thư viện OpenGL
#   pragma comment(lib, "glu32.lib")    // thư viện OpenGL Utility
#   pragma comment(lib, "glew32.lib")   // thư viện GLEW
#   pragma comment(lib, "sdl2.lib")     // thư viện SDL2
#   pragma comment(lib, "sdl2main.lib") // thư viện chứa hàm SDL_main
#endif

#define WND_NAME    "Lab01"             // tiêu đề cửa sổ SDL
#define WND_WIDTH   640                 // chiều rộng cửa sổ SDL (pixel)
```

```
#define WND_HEIGHT  480                 // chiều cao cửa sổ SDL (pixel)

SDL_Window * mainWindow = NULL;         // con trỏ đến cửa sổ SDL
SDL_GLContext context = NULL;           // ngữ cảnh OpenGL

GLboolean appInit() {                   // hàm khởi tạo
    if (SDL_Init(SDL_INIT_VIDEO) < 0) return GL_FALSE;
    mainWindow = SDL_CreateWindow(      // tạo cửa sổ SDL, canh vào giữa màn hình
                    WND_NAME,
                    SDL_WINDOWPOS_CENTERED, SDL_WINDOWPOS_CENTERED,
                    WND_WIDTH, WND_HEIGHT,
                    SDL_WINDOW_OPENGL);
    // tạo ngữ cảnh OpenGL và kết nối vào cửa sổ SDL đã có
    context = SDL_GL_CreateContext(mainWindow);

#ifndef __APPLE__                       // khởi tạo GLEW nếu không phải OSX
    glewExperimental = GL_TRUE;         // Có thể bỏ phần này nếu không sử dụng
    if (GLEW_OK != glewInit())          // các tính năng từ OpenGL 2.0 trở lên
        return GL_FALSE;
#endif

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0, WND_WIDTH, WND_HEIGHT, 0);

    return GL_TRUE;                     // mọi thứ đều ổn
}

void appDraw() {                        // hàm dựng hình
    glClearColor(0, 0, 0, 0);           // xóa buffer về màu đen
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_TRIANGLES);              // vẽ tam giác có tọa độ 3 đỉnh tại
        glVertex2i(320, 160);           // v1 = (320, 160)
        glVertex2i(160, 320);           // v2 = (160, 320)
        glVertex2i(480, 320);           // v3 = (480, 320)
    glEnd();
}

void appExec() {                        // vòng lặp chính
    GLboolean loop = GL_TRUE;
    while (loop) {
        SDL_Event event;
        if(SDL_PollEvent(&event)) {     // lấy sự kiện
            if (event.type == SDL_QUIT || event.key.keysym.sym ==  SDLK_ESCAPE)
                loop = GL_FALSE;        // nhấn ESC = thoát vòng lặp
        }
        appDraw();                      // đặt hàm dựng hình tại đây
        SDL_GL_SwapWindow(mainWindow);  // swap buffers
    }
}

void appShutdown() {                    // hàm dọn dẹp
    SDL_GL_DeleteContext(context);      // hủy bỏ ngữ cảnh OpenGL
```

```
    SDL_DestroyWindow(mainWindow);      // hủy cửa sổ OpenGL
    SDL_Quit();                         // kết thúc phiên làm việc của SDL
}

// Do sử dụng thư viện SDL nên hàm main bắt buộc phải có 2 đối số
int main(int argc, char ** argv) {
    if (GL_FALSE == appInit()) {
        std::cout << "Unable to init SDL\n";
        return -1;
    }
    appExec();
    appShutdown();
    return 0;
}
```

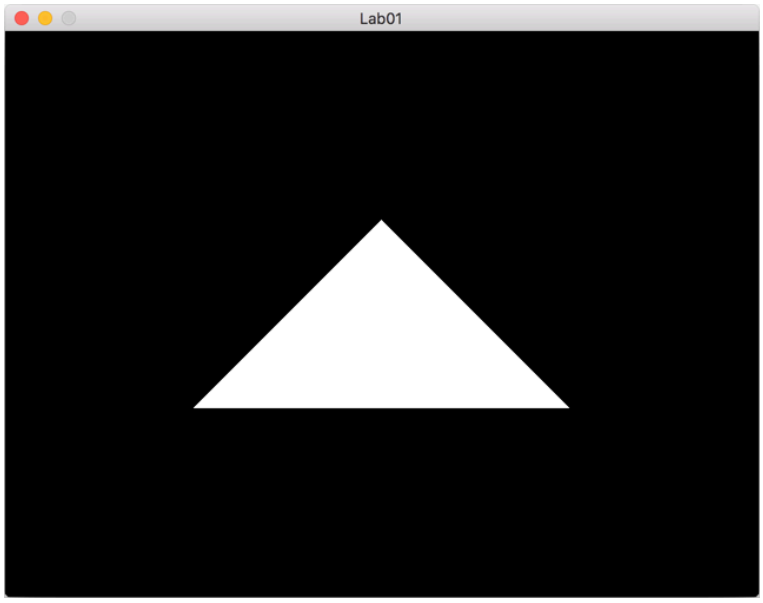
b. Tập tin `MAKEFILE`

```
CXX=g++                         # Trình biên dịch G++
CFLAGS=-Wall -std=c++11         # tùy chọn biên dịch
EXE=                            # phần mở rộng của chương trình
PROJECT=lab01                   # tên của project và tập tin chương trình

# Hỗ trợ biên dịch trên các compiler và hệ điều hành khác nhau
ifeq ($(OS),Windows_NT)         # Môi trường MinGW32
    CFLAGS+=-s
    LIBS=-lmingw32 -lopengl32 -lglu32 -lglew32 -lSDL2main -lSDL2
    LFLAGS=$(LIBS)
    EXE=.exe
    RM=del
else
    UNAME_S := $(shell uname -s)

    ifeq ($(UNAME_S),Linux)     # Môi trường Linux
        LIBS=-lGLU -lGL -lGLEW -lSDL2
        LFLAGS=$(LIBS) `sdl-config --cflags --libs`
        RM=rm -f
    else
    ifeq ($(UNAME_S),Darwin)    # Môi trường OSX
        LIBS=-framework OpenGL -framework SDL2
        LFLAGS=$(LIBS)
        RM=rm -f
    endif
    endif
endif

all: $(PROJECT).cpp
    $(CXX) $(CFLAGS) $< -o $(PROJECT) $(LFLAGS)

clean:
    $(RM) $(PROJECT)$(EXE)
```

Ta nhập nội dung của tập tin `lab01.cpp` vào trình soạn thảo của môi trường phát triển Visual Studio, sau đó biên dịch và chạy thử ứng dụng này. Đối với các lập trình viên

sử dụng MinGW32 hoặc GNU C/C++ trên nền Ubuntu hoặc OSX, tạo một folder có tên `LAB01` trên *Desktop*, nhập toàn bộ nội dung của tập tin `lab01.cpp` và `makefile` và đặt chúng vào trong thư mục này, sau đó sử dụng cửa sổ *PowerShell* trên Windows hoặc *Terminal* trên Ubuntu/OSX để biên dịch bằng các lệnh dưới đây. Tập tin makefile đã cho có hỗ trợ các trình biên dịch trên cả 3 hệ điều hành là Windows, Linux và OSX với trình biên dịch MinGW32, GNU C/C++. Nếu quá trình biên dịch thành công, ta sẽ có tập tin với tên gọi `lab01.exe` hoặc `lab01` được kết xuất trong cùng thư mục, đây chính là tập tin chương trình đã được biên dịch. Để thực hiện ứng dụng này ta chỉ việc nhập tên tập tin chương trình này và *Enter*.

```
$cd ~/Desktop/LAB01          // chuyển vào thư mục LAB01 trên Desktop
$make                        // gọi tiện ích make
$./lab01                     // chạy chương trình đã biên dịch
```



Hình 1 Cửa sổ ứng dụng OpenGL LAB01

Mã nguồn chương trình kèm theo bài tập: `LAB01.ZIP`.

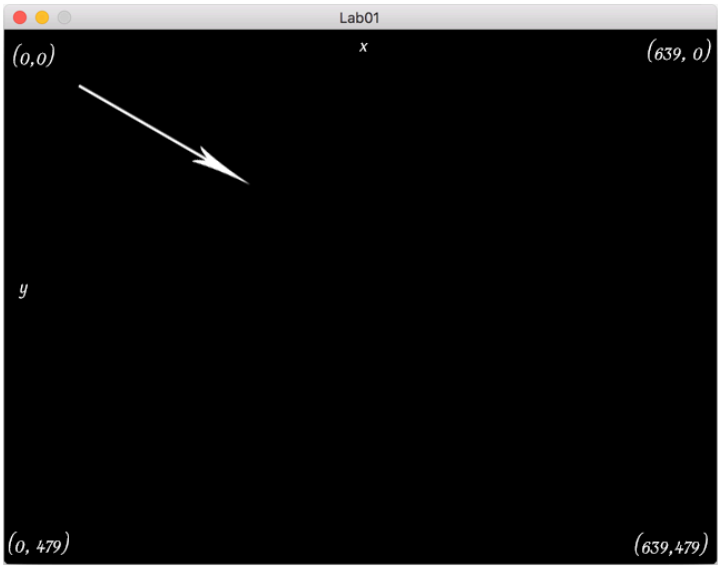
Lưu ý

Khi biên dịch mã nguồn bài tập với Visual Studio, để tránh nhận được thông báo lỗi về *precompile headers*, chèn thêm đoạn mã sau vào đầu tập tin `LAB01.CPP`. Hoặc ta có thể vô hiệu hóa hoàn toàn việc sử dụng *precompile headers* thông qua tùy chọn trong mục *Properties* của project ứng dụng.

```
#include “stdafx.h”          // sử dụng precompile header, chỉ áp dụng với MSVC
```

Ứng dụng này sử dụng phép chiếu trực giao 2D thông qua hàm `gluOrtho2D` để xây dựng hệ trục tọa độ theo biểu diễn tại Hình 2 dựa trên chiều rộng và chiều cao của cửa sổ

ứng dụng. Chiều dương của trục x từ trái sang phải. Chiều dương của trục y hướng từ trên xuống. Hệ trục tọa độ này tương tự hệ trục tọa độ được SDL2 sử dụng.



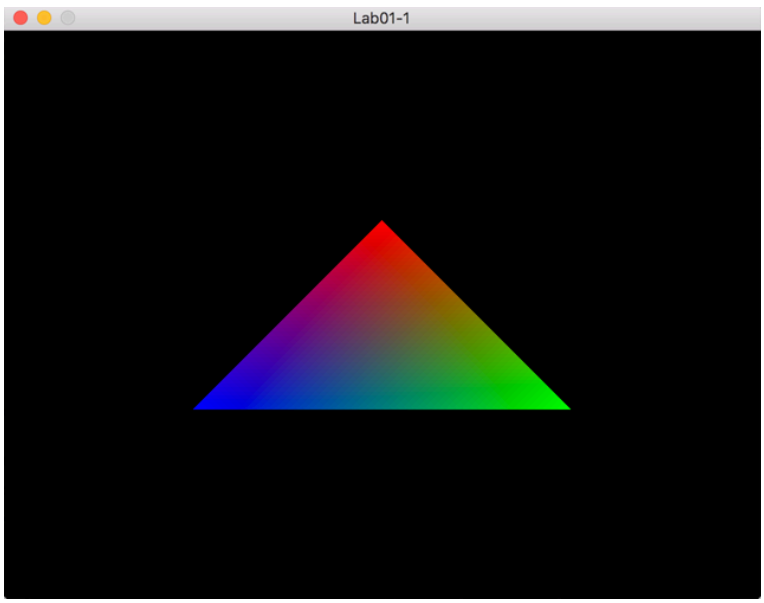
Hình 2. Hệ trục tọa độ của ứng dụng LAB01

Bài tập

Bài 1

Chỉnh sửa hàm `appDraw` sao cho kết quả nhận được là một tam giác RGB với mỗi đỉnh lần lượt có các màu *đỏ*, *lục* và *lam* (Hình 3).

Gợi ý: ta sử dụng hàm `glColor3f` với 3 thành phần `r`, `g`, `b` trong đoạn [0..1] để gán trị màu cho từng đỉnh của tam giác.



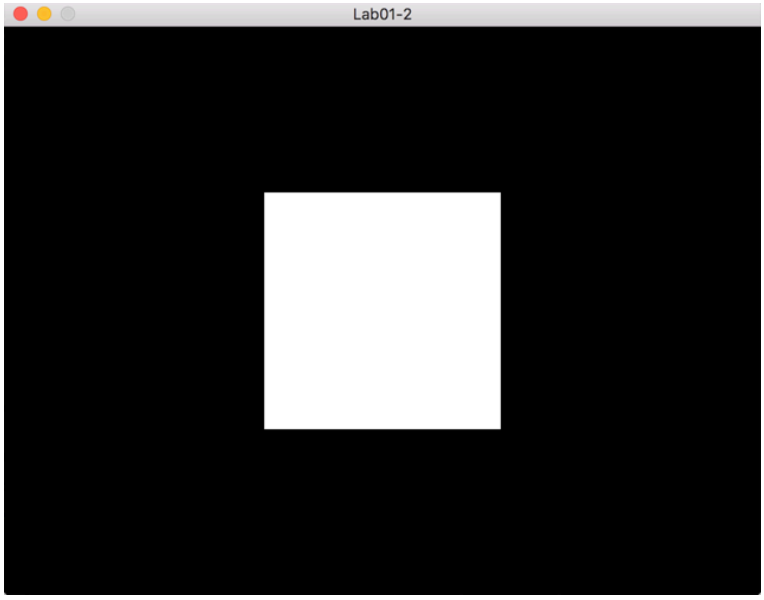
Hình 3. Kết xuất của Bài 1.

Mã nguồn chương trình:

```
void appDraw()
{
    glClearColor(0, 0, 0, 0);            // xóa buffer về màu đen
    glClear(GL_COLOR_BUFFER_BIT);

    glBegin(GL_TRIANGLES);               // vẽ tam giác RGB
        glColor3f(1.0f, 0.0f, 0.0f);     // màu đỏ
        glVertex2i(320, 160);
        glColor3f(0.0f, 1.0f, 0.0f);     // màu xanh lá cây
        glVertex2i(480, 320);
        glColor3f(0.0f, 0.0f, 1.0f);     // màu xanh dương
        glVertex2i(160, 320);
    glEnd();
}
```

Bài 2

Xây dựng một hàm vẽ hình vuông có độ dài cạnh là a bắt đầu từ tọa độ (x, y). Áp dụng để vẽ hình vuông có độ dài cạnh là $a = 200$ và được đặt vào chính giữa cửa sổ OpenGL (Hình 4). Gợi ý: sử dụng `glBegin` và `GL_POLYGON`.



Hình 4. Kết xuất của Bài 2.

Mã nguồn chương trình:

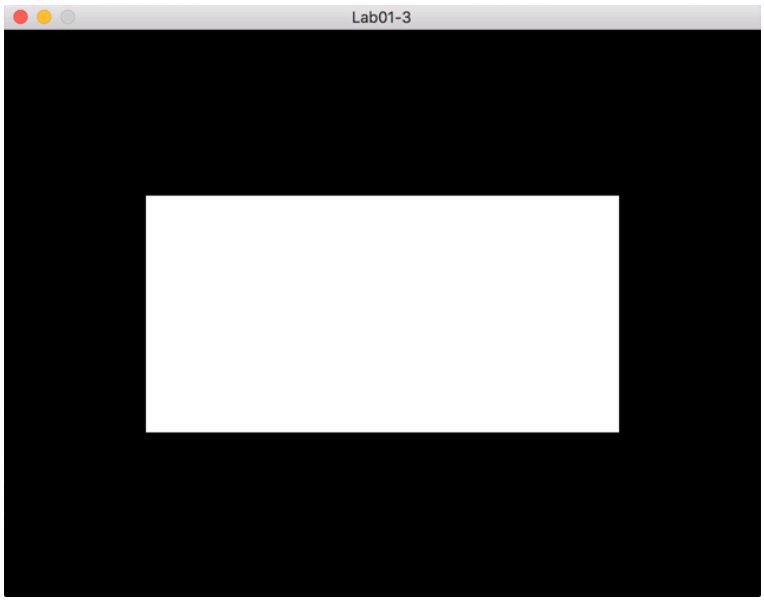
```
void glSquare(GLint x, GLint y, GLint a)
{
    glBegin(GL_POLYGON);
        glVertex2i(x, y);
        glVertex2i(x + a, y);
        glVertex2i(x + a, y + a);
        glVertex2i(x, y + a);
    glEnd();
}
```

```
void appDraw()
{
    glClearColor(0, 0, 0, 0);
    glClear(GL_COLOR_BUFFER_BIT);

    GLint size = 200;                   // độ dài cạnh = 200
    GLint x = (WND_WIDTH-size)/2;       // canh vào giữa theo x
    GLint y = (WND_HEIGHT-size)/2;      // canh vào giữa theo y
    glSquare(x, y, size);
}
```

Bài 3

Tương tự Bài 2, xây dựng một hàm vẽ hình chữ nhật có độ dài chiều rộng là a và chiều cao là b, bắt đầu tại tọa độ (x, y). Áp dụng để vẽ một hình chữ nhật có chiều rộng 400, chiều cao 200 và được đặt vào chính giữa cửa sổ OpenGL (Hình 5).

Gợi ý: sử dụng `glBegin` và tham số `GL_POLYGON`.



Hình 5. Kết xuất của Bài 3.

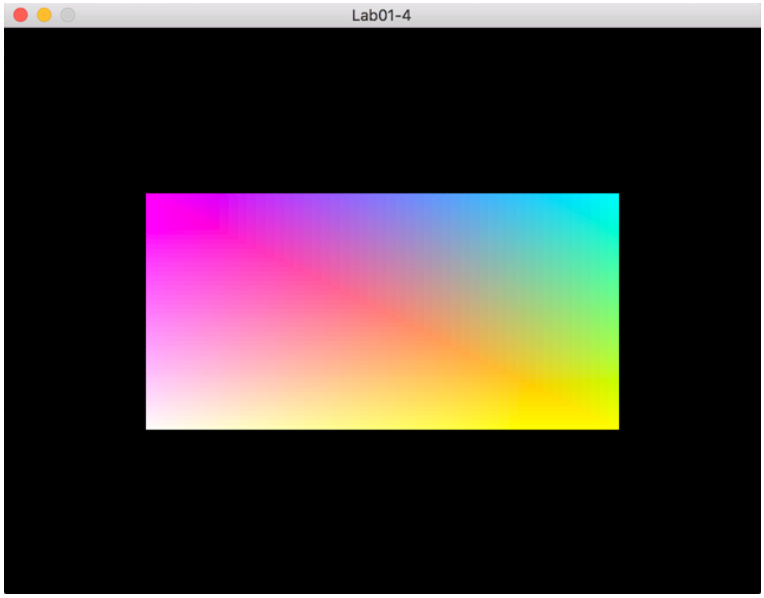
Mã nguồn chương trình:

```
void glRect(GLint x, GLint y, GLint width, GLint height)
{
    glBegin(GL_POLYGON);
        glVertex2i(x, y);                       // (trên, trái)  +------>
        glVertex2i(x + width, y);               // (trên, phải)  ^       |
        glVertex2i(x + width, y + height);      // (dưới, phải)  |       v
        glVertex2i(x, y + height);              // (dưới, trái)  +<------+
    glEnd();
}

void appDraw()
{
    glClearColor(0, 0, 0, 0);
```

```
    glClear(GL_COLOR_BUFFER_BIT);
    GLint w = 400, h = 200;              // chiều rộng = 400, chiều cao = 200
    GLint x = (WND_WIDTH-w)/2;           // canh vào giữa màn hình
    GLint y = (WND_HEIGHT-h)/2;
    glRect(x, y, w, h);
}
```

Bài 4

Áp dụng kỹ thuật ở Bài 1, hãy bổ sung thêm mã hàm `glRect` ở Bài 3 để vẽ ra một hình chữ nhật như biểu diễn tại Hình 6.



Hình 6. Kết xuất của Bài 4.

Mã nguồn chương trình:

```
void glRect(GLint x, GLint y, GLint width, GLint height)
{
    glBegin(GL_POLYGON);
        glColor3f(1.0f, 0.0f, 1.0f);            // đỉnh có màu màu hồng
        glVertex2i(x, y);                       // (trên, bên trái)
        glColor3f(0.0f, 1.0f, 1.0f);            // đỉnh có màu xanh da trời
        glVertex2i(x + width, y);               // (trên, bên phải)
        glColor3f(1.0f, 1.0f, 0.0f);            // đỉnh có màu xanh vàng
        glVertex2i(x + width, y + height);      // (dưới, bên phải)
        glColor3f(1.0f, 1.0f, 1.0f);            // đỉnh có màu trắng
        glVertex2i(x, y + height);              // (dưới, bên trái)
    glEnd();
}
```

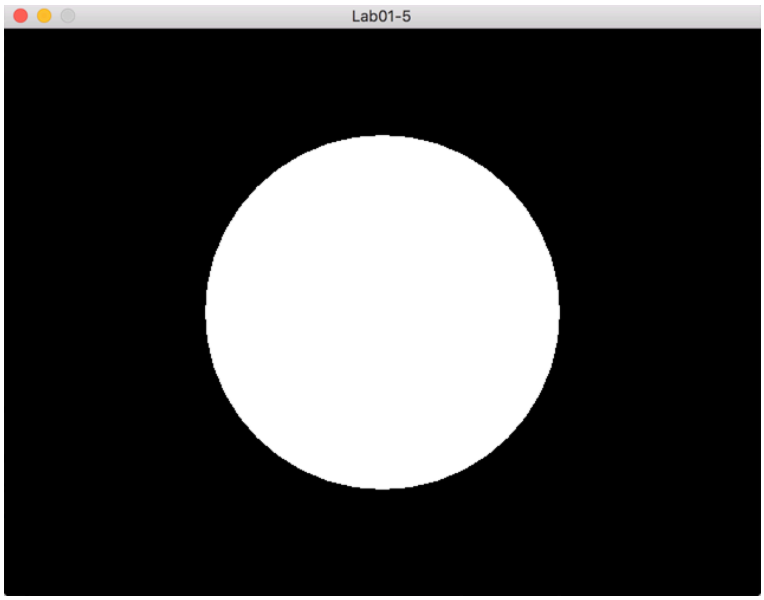
Bài 5*

Sử dụng `glBegin` và `GL_TRIANGLE_FAN`, hãy xây dựng một hàm vẽ hình tròn có tâm (x, y) và bán kính r. Áp dụng vẽ hình tròn có $r = 150$ và tâm tại tọa độ $(320, 240)$ như tại Hình 7.

Mã nguồn chương trình:

```
#define STEPS 40    // số cung trên hình tròn, STEPS càng lớn hình tròn càng mịn.

void glCircle(GLint x, GLint y, GLint radius)
{
    GLfloat twicePi = (GLfloat) 2.0f * M_PI;
    glBegin(GL_TRIANGLE_FAN);
        glVertex2i(x, y);               // tâm hình tròn
        for(int i = 0; i <= STEPS; i++) // lặp STEPS lần, mỗi lần vẽ 1 cung
        {
            glVertex2i( (GLint) (x + (radius * cos(i * twicePi / STEPS))+0.5),
                        (GLint) (y + (radius * sin(i * twicePi / STEPS))+0.5));
        }
    glEnd();
}

void appDraw()
{
    glClearColor(0, 0, 0, 0);
    glClear(GL_COLOR_BUFFER_BIT);
    glCircle(320, 240, 150);      // tâm tại (320, 240), bán kính 150 pixel
}
```

Lưu ý, do có sử dụng thêm hàm `sin`, `cos` và hằng `M_PI` nên ta cần include tập tin `math.h` vào đầu tập tin mã nguồn chương trình.



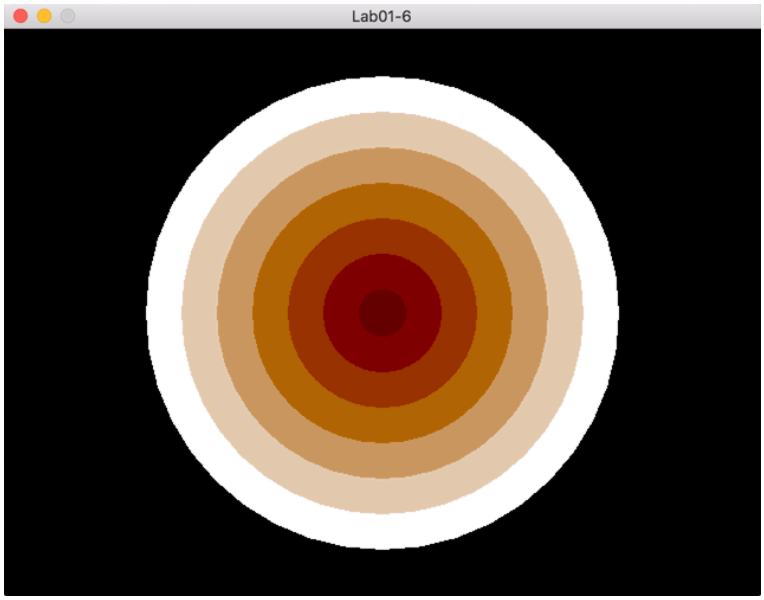
Hình 7. Kết xuất của Bài 5.*

Bài 6

Mở rộng Bài 5*, hãy vẽ các vòng tròn đồng tâm tại điểm $(320, 240)$ như Hình 8. Bán kính ban đầu $r = 100$, giá trị màu ban đầu $(red = 1.0, green = 1.0, blue = 1.0)$. Sau mỗi bước lặp, ta giảm bán kính một lượng là $\Delta r = 30$, $\Delta red = 0.1, \Delta green = 0.2, \Delta blue = 0.3$. Quá trình lặp này dừng khi $r \leq 0$.

Mã nguồn chương trình:

```
void appDraw()
{
    glClearColor(0, 0, 0, 0);
    glClear(GL_COLOR_BUFFER_BIT);

    GLfloat red = 1.0f;                 // giá trị màu R, G, B ban đầu
    GLfloat green = 1.0f;
    GLfloat blue = 1.0f;

    for (int r = 200; r > 0; r -= 30)   // bán kính ban đầu r = 200. Lặp cho đến khi
    {                                   // r <= 0. Sau mỗi bước, r giảm 30 đơn vị
        glColor3f(red, green, blue);
        glCircle(320, 240, r);          // tâm các hình tròn tại (320, 240)
        red -= 0.1f;                    // tính lại các giá trị R, G, B
        green -= 0.2f;                  // theo yêu cầu đề bài
        blue -= 0.3f;
    }
}
```



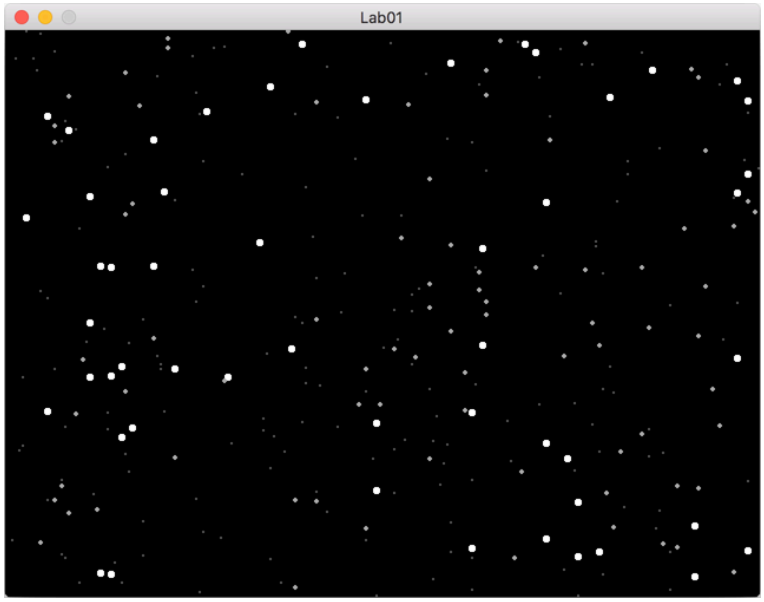
Hình 8. Kết xuất của Bài 6

Bài 7*

Sử dụng hàm `glCircle` đã cài đặt ở Bài 5*, hãy viết ứng dụng đồ họa sử dụng OpenGL mô phỏng hoạt cảnh chuyển động ngang của một bầu trời sao với các yêu cầu sau:

- Tổng số lượng các ngôi sao trên bầu trời là 300.
- Mỗi một ngôi sao ban đầu được phát sinh với các đặc điểm sau:
    - Tọa độ (x, y) của các ngôi sao được phát sinh ngẫu nhiên trong phạm vi kích thước của cửa sổ OpenGL.
    - Bán kính r của các ngôi sao được phát sinh ngẫu nhiên trong khoảng từ 1 đến 3 pixel.

  - Cường độ sáng i của ngôi sao phụ thuộc vào bán kính của ngôi sao đó theo tỷ lệ $i = r/3$.
  - Vận tốc di chuyển ngang của ngôi sao thứ i, ký hiệu Δv_i, được tính dựa trên công thức: $\Delta v_i = 2r_i + v$. Với v là một giá trị ngẫu nhiên trong khoảng từ 1 đến 3.
- Tại bước lặp thứ k, các ngôi sao liên tục di chuyển ngang theo chiều từ trái sang phải dựa trên vận tốc Δv_i của chúng: $x_i^k = x_i^{k-1} + \Delta v_i$.
- Nếu ngôi sao thứ i di chuyển vượt quá chiều rộng cửa sổ, ta phát sinh lại ngẫu nhiên ngôi sao này theo các yêu cầu ở trên, ngoại trừ tọa độ x_i của ngôi sao này bắt đầu từ cạnh trái của cửa sổ: $x_i = 0$, tạo cảm giác một ngôi sao mới di chuyển ngang cửa sổ.



Hình 9. Kết xuất từ Bài 7.*

Mã nguồn chương trình:

```
#define MAX_STARS   300                 // tổng số ngôi sao
typedef struct star                     // cấu trúc một ngôi sao
{
    GLint   x, y;                       // tọa độ x, y
    GLint   radius;                     // bán kính
    GLint   velocity;                   // vận tốc di chuyển
    GLfloat intensity;
} STAR;
STAR sky[MAX_STARS];                    // mảng chứa 300 ngôi sao

void skyInit()                          // phát sinh ban đầu 300 ngôi sao
{
    for (int i = 0; i < MAX_STARS; i++)
    {
        sky[i].x = rand() % WND_WIDTH;  // ngẫu nhiên từ 0..WND_WIDTH-1
        sky[i].y = rand() % WND_HEIGHT; // ngẫu nhiên từ 0..WND_HEIGHT-1
        sky[i].radius = 1 + rand() % 3; // ngẫu nhiên từ 1..3
        sky[i].intensity = sky[i].radius / 3.0f;
```

```
        sky[i].velocity = sky[i].radius*2 + rand() % 3;
    }
}

void skyDraw()                          // tạo hoạt cảnh chuyển động
{
    glClearColor(0, 0, 0, 0);
    glClear(GL_COLOR_BUFFER_BIT);

    for(int i = 0; i < MAX_STARS; i++)  // duyệt qua tất cả các ngôi sao
    {
        // khi r = g = b -> màu trên thang xám
        glColor3f(sky[i].intensity, sky[i].intensity, sky[i].intensity);
        glCircle(sky[i].x, sky[i].y, sky[i].radius);

         // cập nhật vị trí mới dựa trên vận tốc chuyển động
        sky[i].x += sky[i].velocity;

        // phát sinh lại ngôi sao nếu nó vượt quá chiều rộng cửa sổ
        if (sky[i].x >= WND_WIDTH)
        {
            sky[i].x = 0;               // vị trí bắt đầu tại cạnh trái cửa sổ
            sky[i].y = rand() % WND_HEIGHT;
            sky[i].radius = 1 + rand() % 3;
            sky[i].intensity = sky[i].radius / 3.0f;
            sky[i].velocity = sky[i].radius * 2 + rand() % 3;
        }
    }
}

void appExec()                          // vòng lặp chính
{
    GLboolean loop = GL_TRUE;

    // đồng bộ tốc độ hoạt cảnh với thời gian quét màn hình (Vertical Sync)
    SDL_GL_SetSwapInterval(1);
    skyInit();                          // phát sinh bầu trời sao

    while (loop)
    {
        SDL_Event event;
        if(SDL_PollEvent(&event))
        {
            if (event.type == SDL_QUIT || event.key.keysym.sym ==  SDLK_ESCAPE)
                loop = GL_FALSE;
        }
        skyDraw();                      // vẽ và tạo chuyển động cho bầu trời sao
        SDL_GL_SwapWindow(mainWindow);  // swap buffers
    }
}
```

Bài 8:

Chỉnh sửa hàm `skyDraw` trong Bài 7* để đảo ngược chuyển động các ngôi sao theo chiều từ phải sang trái.

Gợi ý:

- Ta giảm tọa độ x của các ngôi sao một lượng là Δv_i. Khi đó công thức tính vị trí của các ngôi sao ở mỗi lần lặp trở thành: $x_i^k = x_i^{k-1} - \Delta v_i$.
- Xét ngôi sao thứ i, sau khi di chuyển hết cạnh trái của cửa sổ, khi đó $x_i < 0$. Ta phát sinh lại ngôi sao này với $x_i = WND\_WIDTH$, tức là ở cạnh phải của cửa sổ OpenGL.

Khi đó, đoạn mã của hàm `skyDraw` trở thành như sau:

```
void skyDraw()                          // tạo hoạt cảnh chuyển động
{
    glClearColor(0, 0, 0, 0);
    glClear(GL_COLOR_BUFFER_BIT);

    for(int i = 0; i < MAX_STARS; i++)
    {
        glColor3f(sky[i].intensity, sky[i].intensity, sky[i].intensity);
        glCircle(sky[i].x, sky[i].y, sky[i].radius);
        sky[i].x -= sky[i].velocity;

        // phát sinh lại ngôi sao nếu nó vượt quá chiều rộng cửa sổ
        if (sky[i].x < 0)
        {
            sky[i].x = WND_WIDTH;
            sky[i].y = rand() % WND_HEIGHT;
            sky[i].radius = 1 + rand() % 3;
            sky[i].intensity = sky[i].radius / 3.0f;
            sky[i].velocity = sky[i].radius * 2 + rand() % 3;
        }
    }
}
```