Manual Instalação e Uso da Canvas2D

Cesar Tadeu Pozzer

26/02/2025

**Uso da Canvas2D**

A Canvas2D nada mais é que uma **API didática** (não é focada em desempenho) feita em cima das API OpenGL/Glut (figura abaixo – mas também roda com a API GLFW) visando facilitar o desenvolvimento de aplicativos gráficos básicos. A FreeGlut é uma API destinada à criação da janela da aplicação e faz o processamento dos eventos de mouse/teclado/renderização. A API OpenGL não oferece comandos para essas atividades que a FreeGlut desempenha.

Veja o Demo da Canvas2D com a GLFW no material da disciplina. A API GLFW é mais moderna e possui atualizações periódicas. A maior desvantagem dela é que não suporta impressão de texto.

OpenGL

FreeGlut - GLFW

Aplicação do usuário

Placa de Vídeo

Canvas2D

O papel da API Canvas2D é simplesmente tornar mais transparente o desenho de primitivas básicas, sem que seja necessário conhecimento do funcionamento das APIs FreeGlut e OpenGL. Ela disponibiliza comandos para desenho de linhas, quadrados e pontos. Também permite a seleção de cores e impressão de textos. Pode ser usada para programar jogos 2D como Space invaders, Tetris, e muitos outros. Pode também ser usada para criação de aplicativos 3D e manipulação de imagens, porém com muito baixa performance.

Por ser baseada no OpenGL, toda programação é baseada em eventos.

|  |
| --- |
| Basicamente, na Canvas2D é feito um bypass dos comandos de teclado e mouse, implementados por meio de callbacks (na FreeGlut isso é feito assim, na GLFW pode ser feito de forma diferente). Esses comandos são agrupados em apenas duas funções, com mais parâmetros. A Canvas2D também faz a inicialização do OpenGL, definição da forma de renderização, que nesse caso é em 2D, e especificação de primitivas gráficas simples usando a sintaxe do OpenGL. |

A Canvas2D é fundamentada em três funções:

1. mouse() – Trata todos os eventos de mouse. A função é chamada quando ocorrer algum evento de mouse, seja movimento, arrasto, clique ou scrool.
2. keyboard() – Trata todos os eventos de teclado. A função é chamada quando ocorrer algum evento de teclado (click ou release).
3. render() – Único local onde devem ser colocados comandos para desenho de primitivas gráficas. Essa função é chamada continuamente. Dependendo da placa de vídeo e das configurações de v-sync, isso pode ocorrer mais de 1000 vezes por segundo.

A forma geral de uso da API é por meio de variáveis globais. Como exemplo, considere a implementação de uma entidade que se move controlada pelo teclado (como no jogo Pac-man). Na função keyboard() devem ser tratados os eventos das setas direcionais que movem a entidade na vertical e horizontal. O seguinte **pseudocódigo** ilustra como deve ser tratado o evento para mover a entidade para a direita.

//variável global que controla a movimentação para a direita

bool g\_direita = false;

void render()

{

if (g\_direita == true)

entidade.x++; //move a entidade para a direita

desenhaEntidade();

}

void keyboard(int tecla)

{

if (tecla == seta\_direita)

g\_direita = true; //muda o estado da variável global

else

g\_direita = false; //para de andar para a direita

}

Deve-se atentar para não colocar laços de repetição verificando eventos dentro da função render(), como no seguinte exemplo. Isso iria fazer a aplicação **travar**.

void render()

{

while (tecla == seta\_direita)

entidade.x++;

}

Pode-se usar laços para fazer o desenho de várias entidades, como no seguinte exemplo.

void render()

{

for (linhas)

for (colunas)

desenhaEntidade(linha, coluna);

}

Para fazer o gerenciamento do que desenhar na Canvas2D pode-se adicionar testes condicionais, como no seguinte exemplo, também fazendo uso de variáveis globais.

void render()

{

if (menu == true)

desenhaMenu();

if (morreu == true)

desenhaTelaFimJogo();

if (jogando == true)

desenhaCenarioJogo();

}

Todos os métodos da Canvas2D são estáticos. Seguem exemplos de uso:

CV::color(1);

CV::translate(100, 100);

CV::point(10, 10); //desenha na posição 110, 110.

CV::rectFill(Vector2(0, 0), Vector2(30, 30));

CV::text(10, 300, “Ola mundo”);

O método CV::translate() simplifica o cálculo de coordenadas para desenho de muitos elementos relacionados entre si. Observe que o este comando não é cumulativo, ou seja, chamar translate(10,10), seguido de translate(20,20), não move a origem para coordenada (30,30), mas sim para a coordenada (20,20). Tudo o que vier a ser desenhado após o translate (funções point(), circle(), rect(), etc) sofrerá uma translação de (20,20). **O CV::translate() deve ser chamado dentro da render()**.

Os seguintes exemplos ilustram o uso de CV::translate(). Como um círculo gerado por coordenadas polares tem coordenadas positivas e negativas, sem uma translação apenas ¼ dele apareceria na tela.

|  |  |
| --- | --- |
| Com | Sem |
| CV::translate(200, 200);  for(ang = 0; ang < 2\*PI; ang += 0.01)  {  x = cos(ang) \* 50;  y = sin(ang) \* 50;  CV::point(x, y);  } | for(ang = 0; ang < 2\*PI; ang += 0.01)  {  x = cos(ang) \* 50;  y = sin(ang) \* 50;  CV::point(200 + x, 200 + y);  } |

A inicialização da Canvas2D é bem simples:

int screenWidth = 500, screenHeight = 500; //largura e altura inicial da canvas2D

int main(void)

{

CV::init(&screenWidth, &screenHeight, "Trabalho 1 da Maria");

CV::run();

}

**Configuração da Canvas2D no Code::Blocks**

Para compilar/executar um aplicativo em OpenGL é necessário a existência de arquivos h, lib e dll. Todos os arquivos necessários para rodar em Windows já estão inseridos e configurados no arquivo de projeto (**canvasGlut.cbp**). Esse arquivo pode ser visualizado em qualquer editor de texto.

<Linker>

<Add library="../lib/libfreeglut32.a" />

<Add library="../lib/libopengl32.a"/>

<Add library="../lib/libglu32.a"/>

</Linker>

Para rodar a Canvas2D, necessita-se de duas coisas:

1. Compilador MinGW 32 bits,
2. IDE do Code::Blocks 32 ou 64 bits.

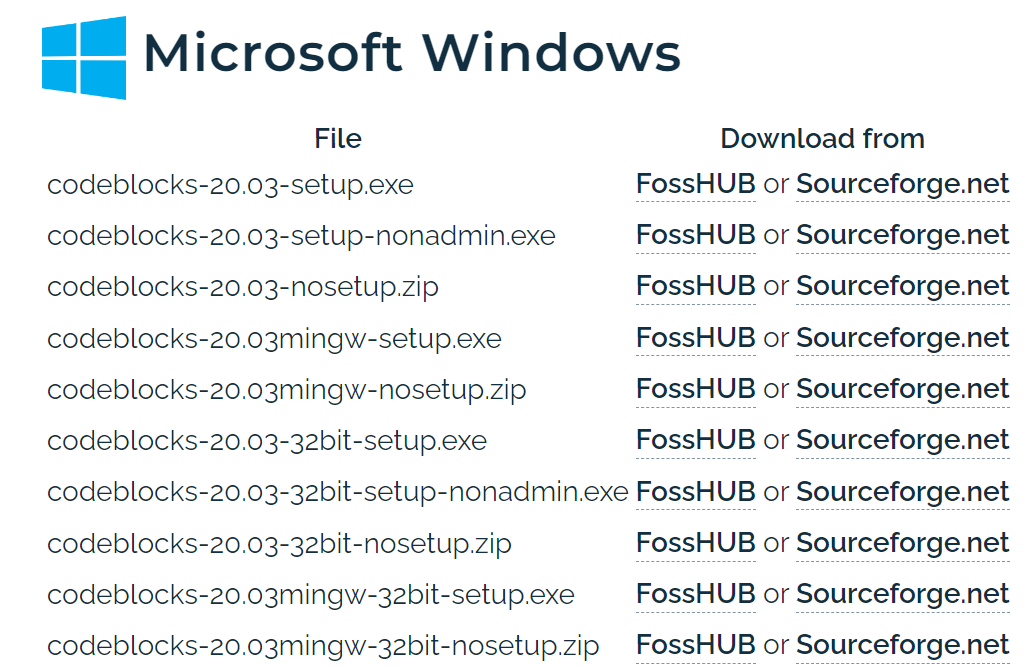
Por questões de compatibilidade de bibliotecas e dlls, deve-se fazer o download da versão do compilador MinGW **32 bits** (visto que as bibliotecas OpenGL são 32 bits).

Quanto a IDE Code::Blocks, existem 3 opções:

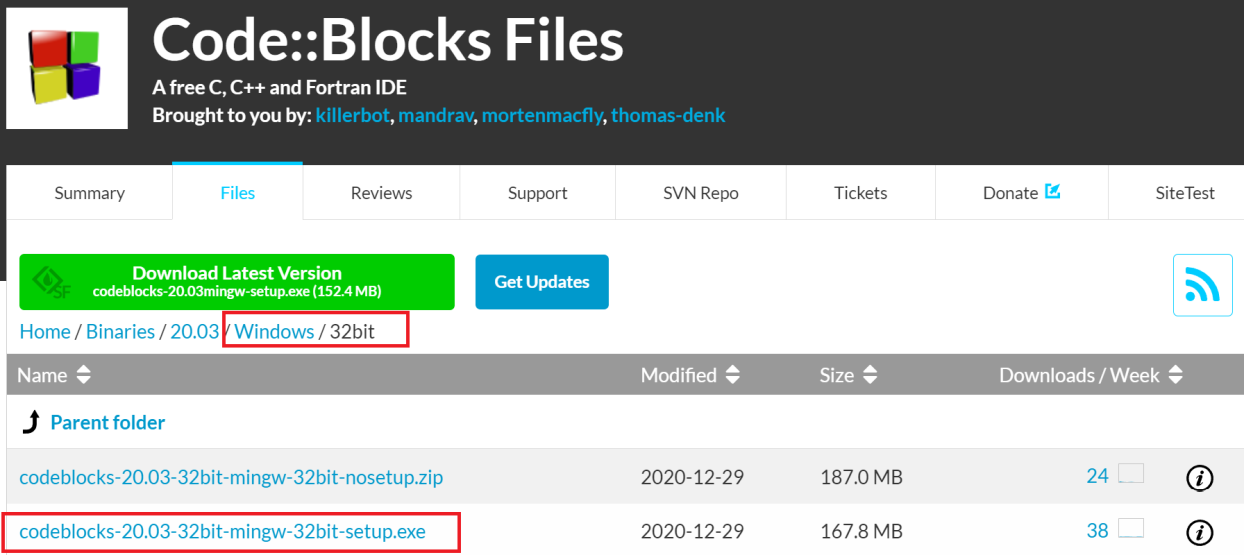
1. Baixar tudo na Versão 17.12, em 32 bits: **codeblocks-17.12mingw-setup.exe**. Basta instalar e está tudo pronto. Já vem no pacote o compilador MinGW 32 bits. (A opção mais fácil)
2. Pode-se baixar tudo na versão 20.03, em 32 bits: **codeblocks-20.03-32bit-mingw-32bit-setup.exe**. Deve-se configurar manualmente o Toolchain executables, como mostrado a seguir. E a interface gráfica da IDE é de baixa qualidade (Não recomendado para telas com alta definição). O compilador MinGW é mais atualizado que a versão 17.12.
3. Pode-se baixar a **IDE** na versão 20.03, em 64 bits: **codeblocks-20.03-setup.exe**. Deve-se também baixar o MinGW 32, que pode ser o da opção 1 (mais velho) ou da opção 2 (mais atual - Melhor opção de todas).

Onde baixar:

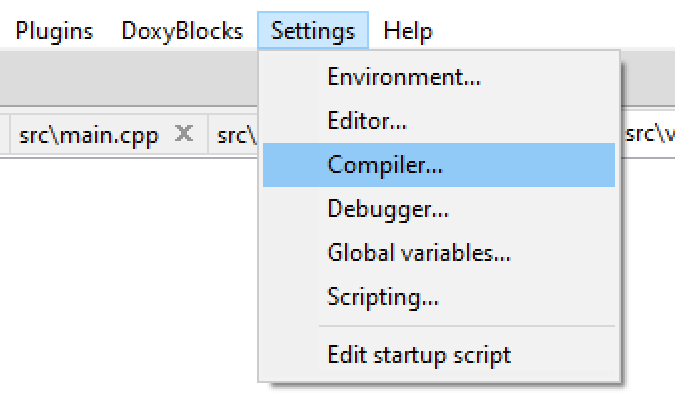
<https://www.codeblocks.org/downloads/binaries/>

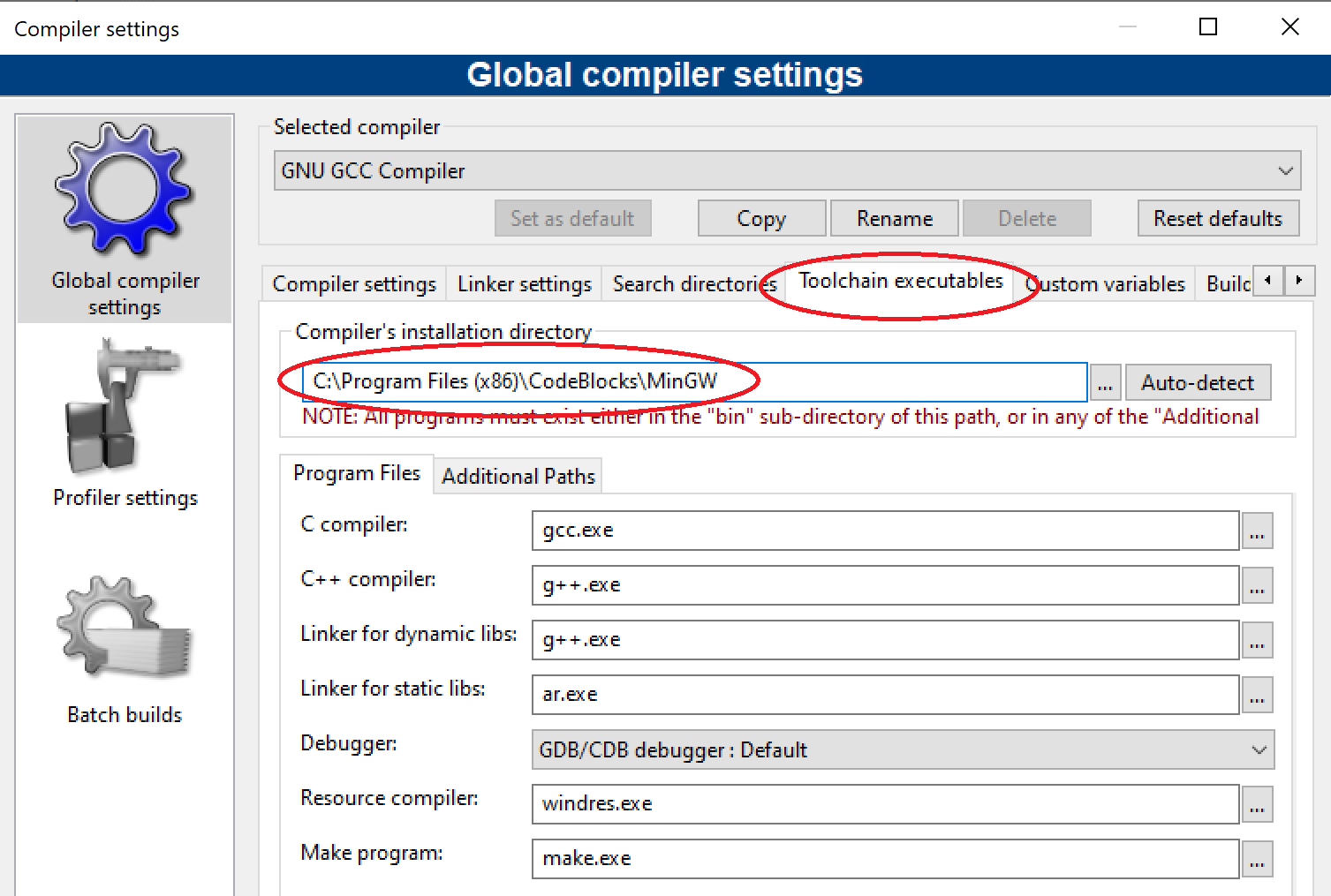


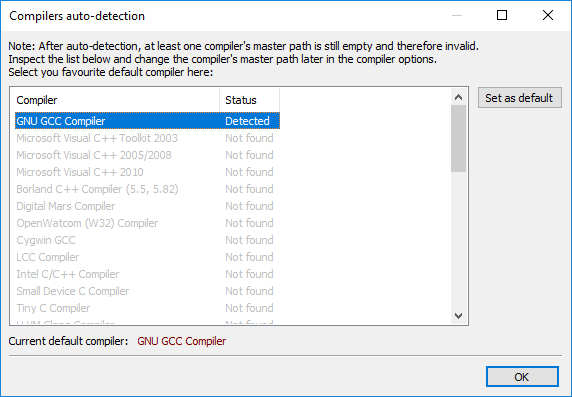




Após a instalação, deve-se certificar que o Code::Blocks esteja utilizando a versão do compilador MinGW correta (na pasta ...x86), independente da versão, conforme ilustrado abaixo:

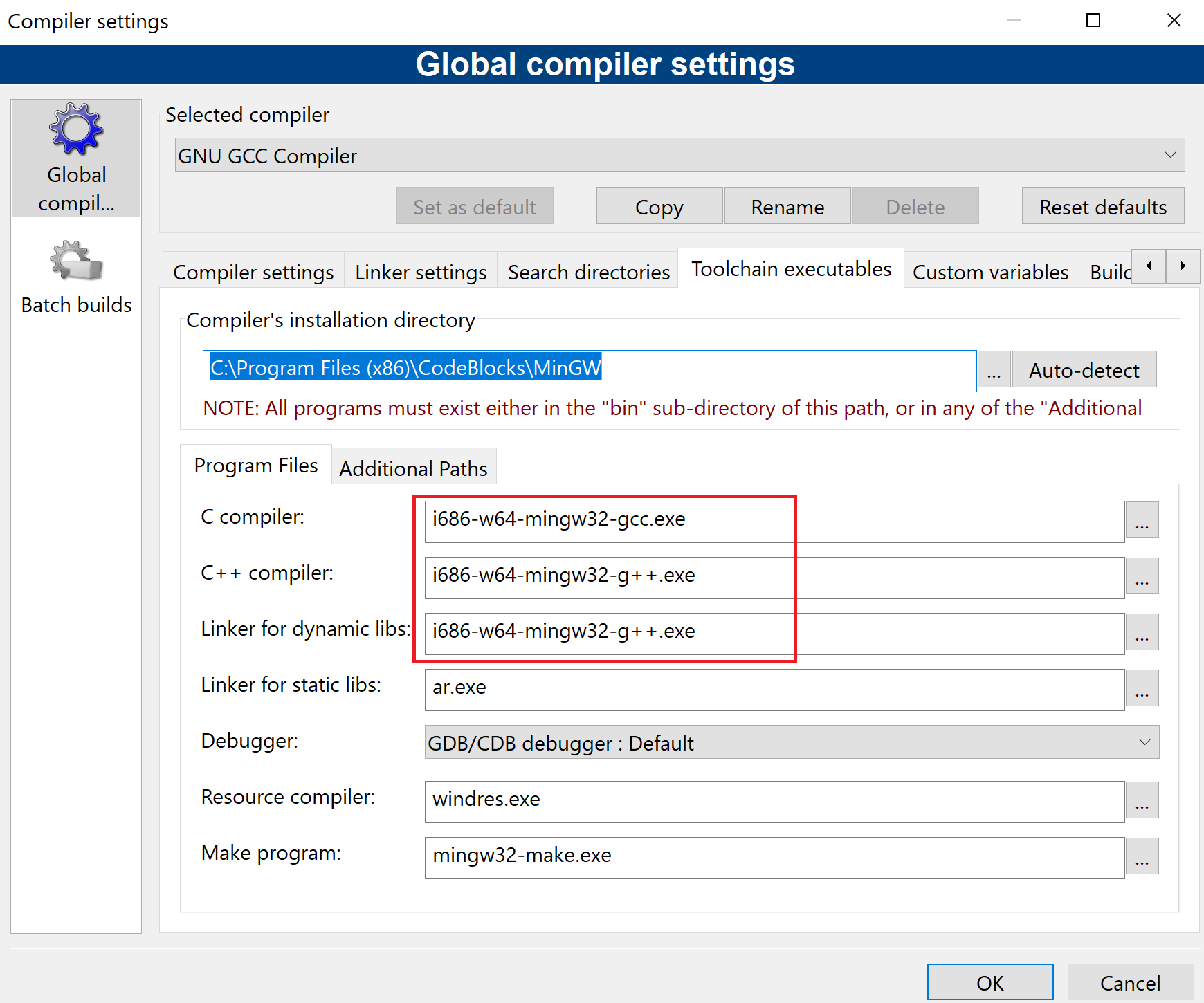






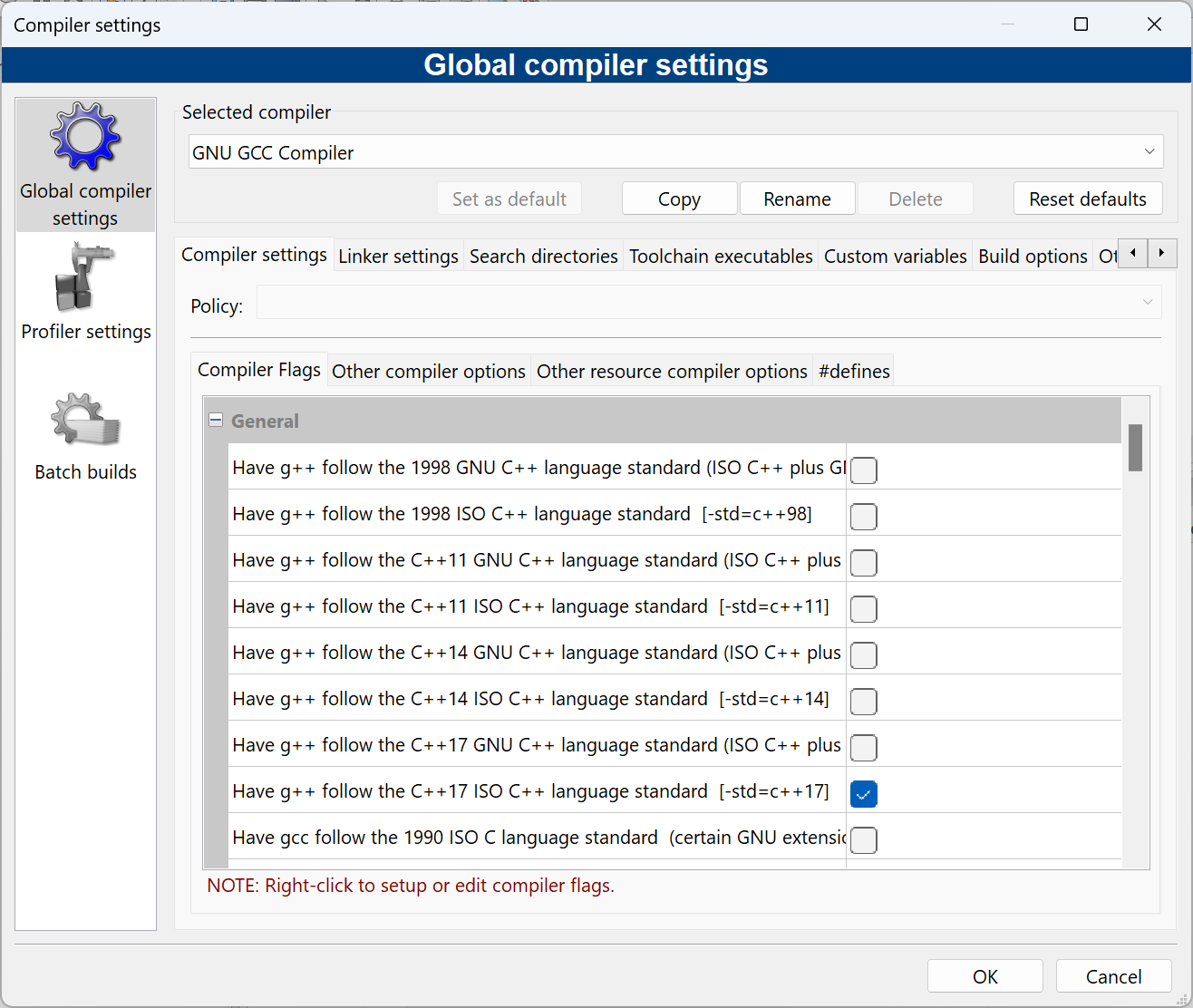
Para uma melhor visualização gráfica da interface do Code::Blocks, recomendo baixar o Code::Blocks 20.03, em 64 bits e vincular ao compilador MinGW da versão 32 bits (Versão 17.12 ou 20.03). Neste caso tem que fazer duas instalações.

Se a IDE for a versão 20.03 em 32 bits (não recomendo), deve-se setar os programas na mão. O Auto-detect não funciona. Abaixo imagem da configuração.



Algumas vezes o Code::Blocks tem problemas na hora de fazer uma build por causa de arquivos de builds antigas. Procurem usar o comando Rebuild ­ do Code::Blocks quando isso acontecer.

Alguns programas podem requerer versões mais atuais da linguagem C++. Para ajustar:



Pode-se também alterar diretamente no arquivo de configuração do projeto:

<Target title="Release">

<Option output="../\_\_bin/Release/canvas" prefix\_auto="1" extension\_auto="1" />

<Option working\_dir="../" />

<Option object\_output="../\_\_obj/Release/" />

<Option type="1" />

<Option compiler="gcc" />

<Compiler>

<Add option="-std=c++11" />

<Add option="-O2 -Wall" />

<Add directory="include" />

</Compiler>

<Linker>

<Add option="-s" />

</Linker>

</Target>

**Canvas2D em outras plataformas**

A forma mais fácil de rodar a canvas2D e o Code::Blocks em outros sistemas operacionais é via a ferramenta Virtual Box: <https://www.virtualbox.org/>.



Pode-se instalar máquinas virtuais para Linux, OS X e Windows. Segue link de um vídeo que ilustra como instalar: <https://www.youtube.com/watch?v=hxbcIEFv34k>

Após instalado o VB, basta instalar o Code::Blocks na versão desejada.

**Configuração para Linux (Por Bruno Torres)**

Seguem algumas dicas para as distribuições do Linux derivadas do Debian ou Ubuntu.

--- Instalação de pacotes

Primeiramente, é preciso ter instalado o gcc/g++, OpenGL e FreeGLUT:

sudo apt-get install build-essential mesa-utils freeglut3-dev

Instalação do Code::Blocks:

sudo apt-get install codeblocks

Caso o pacote não seja encontrado, dê uma olhada neste link.

--- Alterações nos projetos do Code::Blocks

É preciso alterar algumas linhas nos projetos do CB para compilar no Linux. É só encontrar os arquivos com final .cbp e editar em qualquer editor de texto.

Você pode substituir as linhas ou apenas comentá-las (recomendado) usando <!-- e -->.

Original (Windows):

<Linker>

<Add library="..\lib\libglu32.a" />

<Add library="..\lib\libopengl32.a" />

<Add library="..\lib\freeglut.lib" />

</Linker>

No Linux, substituir por:

<Linker>

<Add option="-lGL" />

<Add option="-lGLU" />

<Add option="-lglut" />

</Linker>

Alguns projetos (e.g. demo gl\_14\_texture) também requerem que as linhas

<Add directory="include" />

dentro das tags <Compiler> sejam removidas ou comentadas. Caso contrário, o compilador irá tentar utilizar os headers que vem com os demos, que são para Windows.