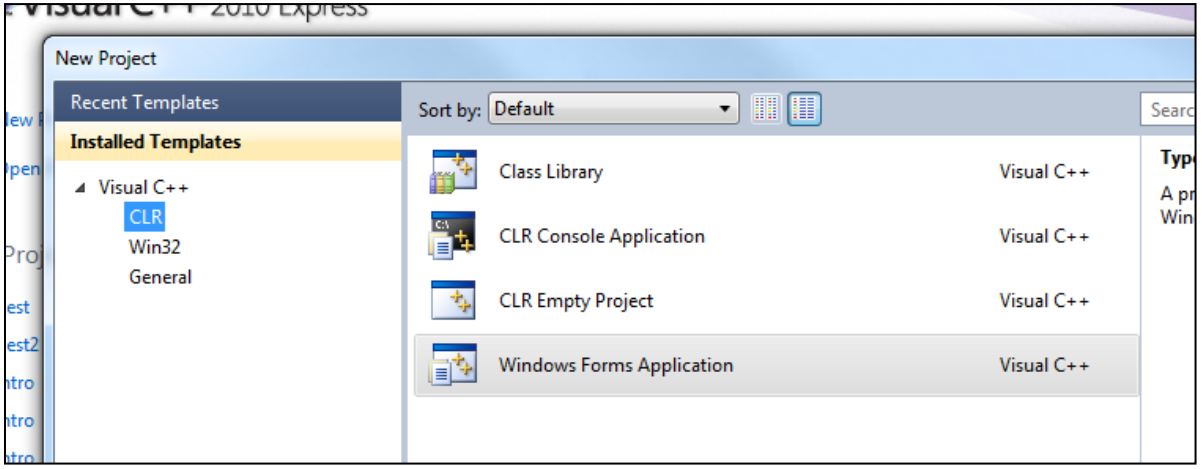
Chapitre 1

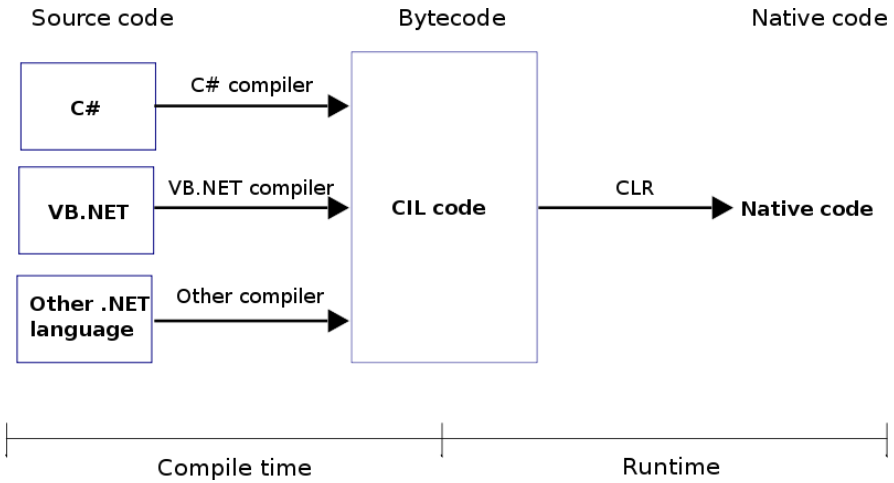
Présentation et introduction à l'infographie 3D

Premier exemple – 2D.

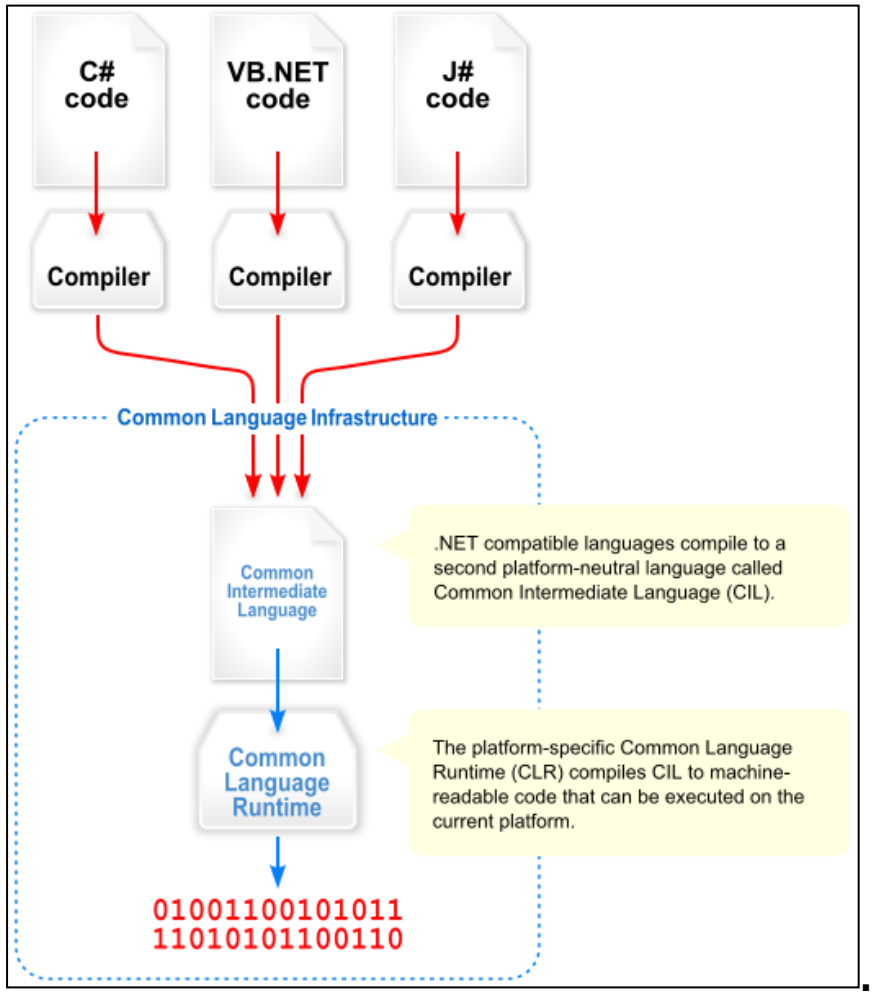
Nous allons utiliser un formulaire CLR.



CLR : Common Language Runtime. C'est la machine virtuelle de Microsoft qui supporte .NET. On peut aussi le voir comme l'implémentation MS de CLI.



CLI : Common Language Infrastructure. C'est la spécification qui décri le code exécutable et le « runtime » du .NET. Cette spécification à été acceptée par ISO.



Votre nouveau projet (le miens se nomme « Test ») contient plusieurs fichiers générés. Pour l'instant, intéressons-nous à Test.cpp et Form1.h.

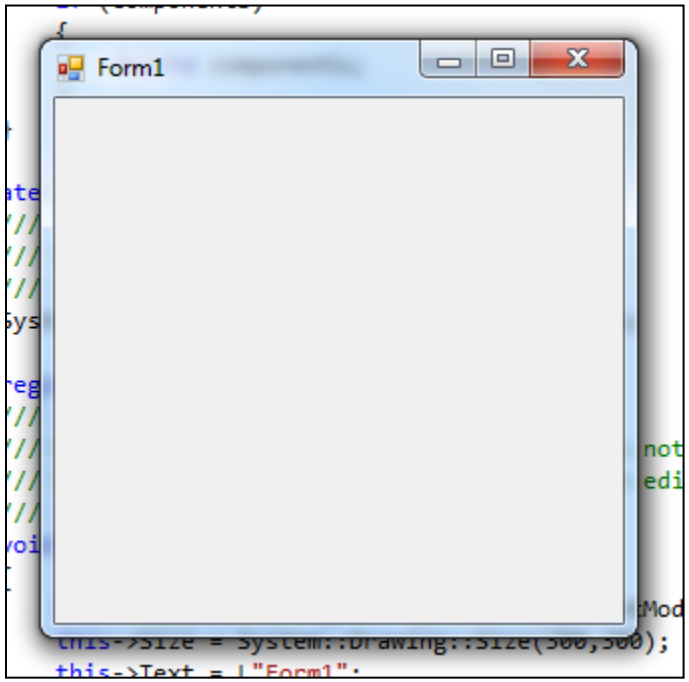
```
1  // Test.cpp : main project file.
2
3  #include "stdafx.h"
4  #include "Form1.h"
5
6  using namespace Test;
7
8  [STAThreadAttribute]
9  int main(array<System::String ^> ^args)
10 {
11     // Enabling Windows XP visual effects before any controls are created
12     Application::EnableVisualStyles();
13     Application::SetCompatibleTextRenderingDefault(false);
14
15     // Create the main window and run it
16     Application::Run(gcnew Form1());
17     return 0;
18 }
19
```

Test.cpp est le point de départ de notre application (fonction main()). Notez la syntaxe du tableau d'arguments. CLR introduit un nouveau type de référence : les « handles », symbole : « ^ ». Nous y reviendrons.

Le fonction « main() » Crée une instance de « Form1 » (notez le « gcnew » qui est utilisé ici) et démarre notre application.

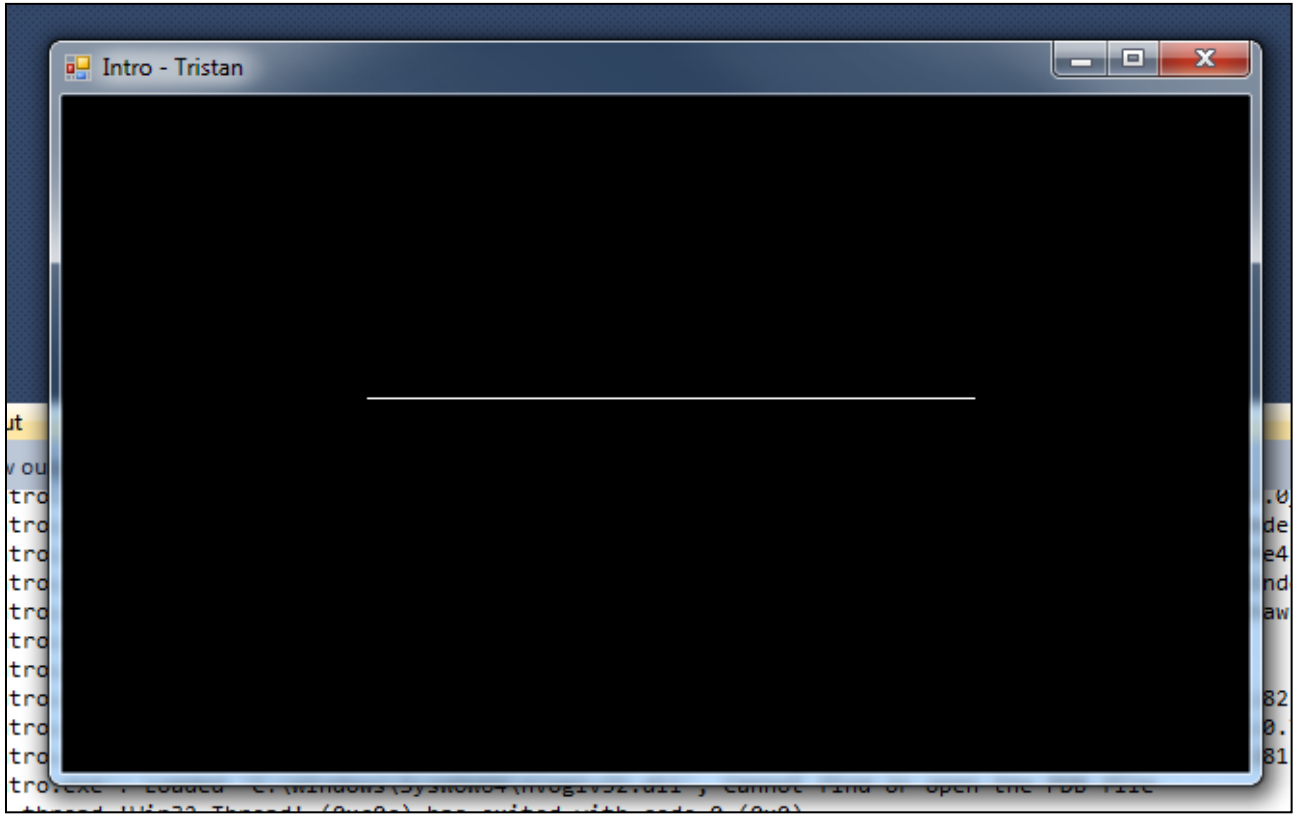
Le fichier Form1.h contient tout le code nécessaire pour définir un formulaire vide. Le studio offre un outil de design, une section du code lui est donc réservée au bas du fichier. Prenez une minute ou deux pour lire le fichier. Vous devriez vous y retrouver assez vite.

À l'exécution, nous obtenons un formulaire vide.



UN PREMIER EXEMPLE

Nous avons un formulaire vide, ajoutons une surface de rendu OpenGL dans laquelle nous allons dessiner une simple ligne blanche.

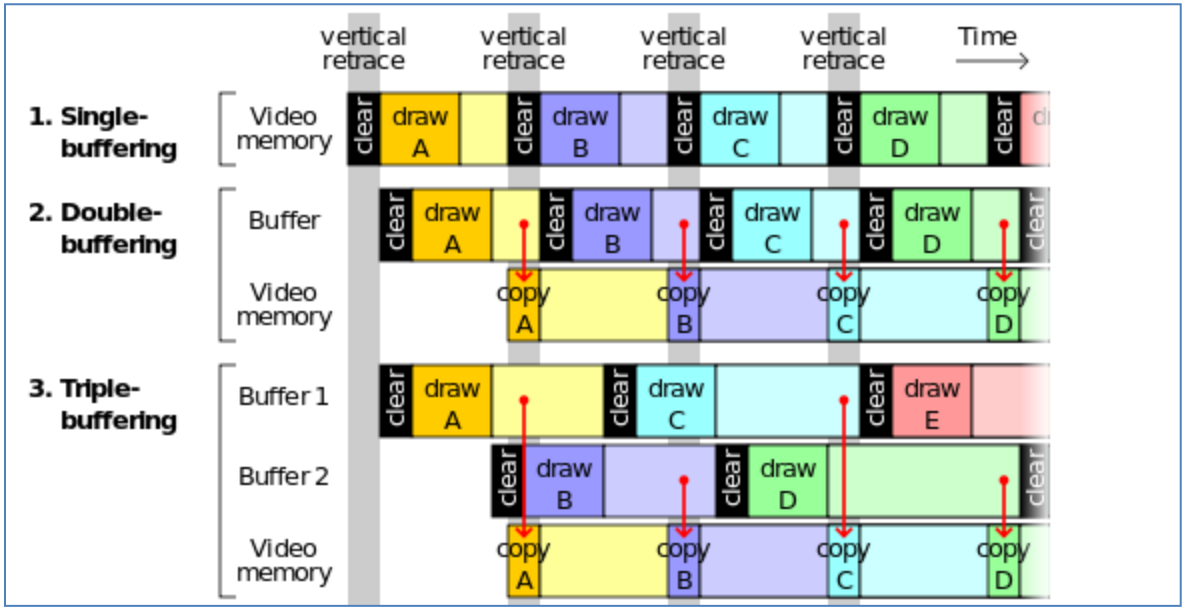


Avant de se lancer dans le code, quelques concepts d'infographie et terme spécifique à OpenGL importants :

- **La scène est le monde virtuel 3D que nous voulons rendre.**
- **Le « viewport » est la surface de rendu que nous donnons à OpenGL. On peut le voir comme les pixels physiques de l'écran pris en charge par le matériel graphique (GPU).**
- **Une matrice est une entité mathématique qui représente une transformation géométrique.**
  - **La matrice « GL_PROJECTION » est responsable de la conversion en 2D de notre vue 3D (c'est la projection !).**
  - **La matrice « GL_MODELVIEW » contient les modifications relative au point de vue et aux transformations au model (déplacements, rotation, etc..).**
  - **Une matrice identité est une matrice « vide », qui ne transforme rien.**
  - **Il y en a d'autres, nous y reviendrons.**
- **Le « Double Buffering » est une technique qui consiste à rendre (dessiner) sur une surface cachée en mémoire et de simplement remplacer l'image à l'écran une fois quelle est complète. L'objectif est de réduire le scintillement. La méthode « SwapBuffers() » est responsable de cette opération.**

- **Nous allons voir qu'OpenGL utilise plusieurs tampons (buffers) pour rendre des scènes. Les plus importants pour l'instant :**
  - **GL_COLOR_BUFFER_BIT : ce sont nos pixels, ce que nous sommes en train de dessiner;**
  - **GL_DEPTH_BUFFER_BIT : une cache de profondeur. Nous ne voulons pas écraser des pixels « plus loin ».**
- **RGBA : un format de pixel 32 bits qui correspond a un octet pour le rouge, le vert, le bleu et l'alpha (transparence).**
- **Vertex : Un point 3D dans l'espace qui est à la base de nos maillages (meshs). C'est le bloc de construction de nos triangles Ili peut avoir plusieurs propriétés comme la couleur, une normal, un poids pour l'animation (skinning), etc…**
- **La projection peut être :**
  - **orthographique « glOrtho »;**
  - **Perspective « gluPerspective »;**
  - **Il y en a d'autres.**

Utilisation de tampons pour le rendu.



Notre fichier « Form1.h » doit prendre cette forme :

```
1  #pragma once
2
3  #include <windows.h>
4  #include <GL/gl.h>
5
6  namespace Intro {
7
8      using namespace System;
9      using namespace System::Windows::Forms;
10
11     public ref class Form1 : public System::Windows::Forms::Form
12     {
13     public:
14         Form1(void);
15
16     protected:
17         ~Form1() {}
18
19         virtual void OnShown(EventArgs^ e) override;
20         virtual Void OnPaint(PaintEventArgs^ e) override;
21         virtual Void OnPaintBackground(PaintEventArgs^ e) override;
22         virtual Void OnResize(EventArgs^ e) override;
23
24     private:
25         Void Init(Void);
26         Void Render(Void);
27         GLvoid ReSizeGLScene(GLsizei width, GLsizei height);
28         GLint MySetPixelFormat(HDC hdc);
29
30     private :
31         HDC m_hDC;
32     };
33 }
```

On note 4 méthodes virtuelles et 4 méthodes privées nécessaires à l'initialisation de notre scène et de notre surface de rendu. Il y a aussi une « handle » de Windows qui sert à communiquer avec notre fenêtre.

Le constructeur initialise le formulaire Windows.

```
7   Form1::Form1(void)
8   {
9       m_hDC = 0;
10
11      SuspendLayout();
12      AutoScaleDimensions = System::Drawing::SizeF(6, 13);
13      AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;
14      ClientSize = System::Drawing::Size(795, 489);
15      Name = L"Form1";
16      Text = L"Intro - Tristan";
17      ResumeLayout(false);
18      SetStyle(ControlStyles::ResizeRedraw, true);
19  }
```

La méthode « OnShown » est appelée juste avant le premier affichage, c'est le bon moment pour initialiser notre surface.

```
21  void Form1::OnShown(EventArgs^ e)
22  {
23      Init();
24      Render();
25  }
```

Nous désactivons les rendus normaux de Windows.

```
Void Form1::OnPaint(PaintEventArgs^ e)
{
    // Désactive le "OnPaint"
    //__super::OnPaint(e);
}

Void Form1::OnPaintBackground(PaintEventArgs^ e)
{
    // Désactive le "OnPaintBackground"
    //__super::OnPaintBackground(e);
}
```

Il faut que la librairie OpenGL soit avertie si la fenêtre est redimensionnée.

```
39  Void Form1::OnResize(EventArgs^ e)
40  {
41      __super::OnResize(e);
42      ReSizeGLScene(Width, Height);
43  }
```

Le « resize » est le bon moment pour donner à OpenGL la taille du « viewport » et les informations de projection. Notez aussi de quelle façon nous activons les différentes matrices (et dans ce cas-ci, nous les réinitialisons)

```
44
45  GLvoid Form1::ReSizeGLScene(GLsizei width, GLsizei height)
46  {
47      glViewport(0,0,width,height);                       // Re-initialise le Viewport
48
49      glMatrixMode(GL_PROJECTION);                        // On active la matrice de projection
50      glLoadIdentity();                                   // La matrice de projection est remise à zero
51
52      glOrtho(-2.0 ,2.0, -2.0 ,2.0, -2.0 ,2.0);
53
54      glMatrixMode(GL_MODELVIEW);                         // On active la matrice modèle / vue
55      glLoadIdentity();                                   // La matrice modèle / vue est remise à zero
56
57      Render();
58  }
59
```

L'initialisation d'OpenGL

```
60  System::Void Form1::Init(System::Void)
61  {
62      m_hDC = GetDC((HWND)this->Handle.ToPointer());
63      if(m_hDC)
64      {
65          MySetPixelFormat(m_hDC);
66          ReSizeGLScene(Width, Height);
67
68          glClearColor(0.f, 0.f, 0.f, 1.f);               // Fond noir
69          glClearDepth(1.f);                              // On vide le tampon de profondeur
70          glEnable(GL_DEPTH_TEST);                        // On active le test de profondeur
71          glDepthFunc(GL_LEQUAL);                         // Type de test
72      }
73  }
74
```

Je vous laisse regarder « MySetPixelFormat » qui n'est pas très intéressante.

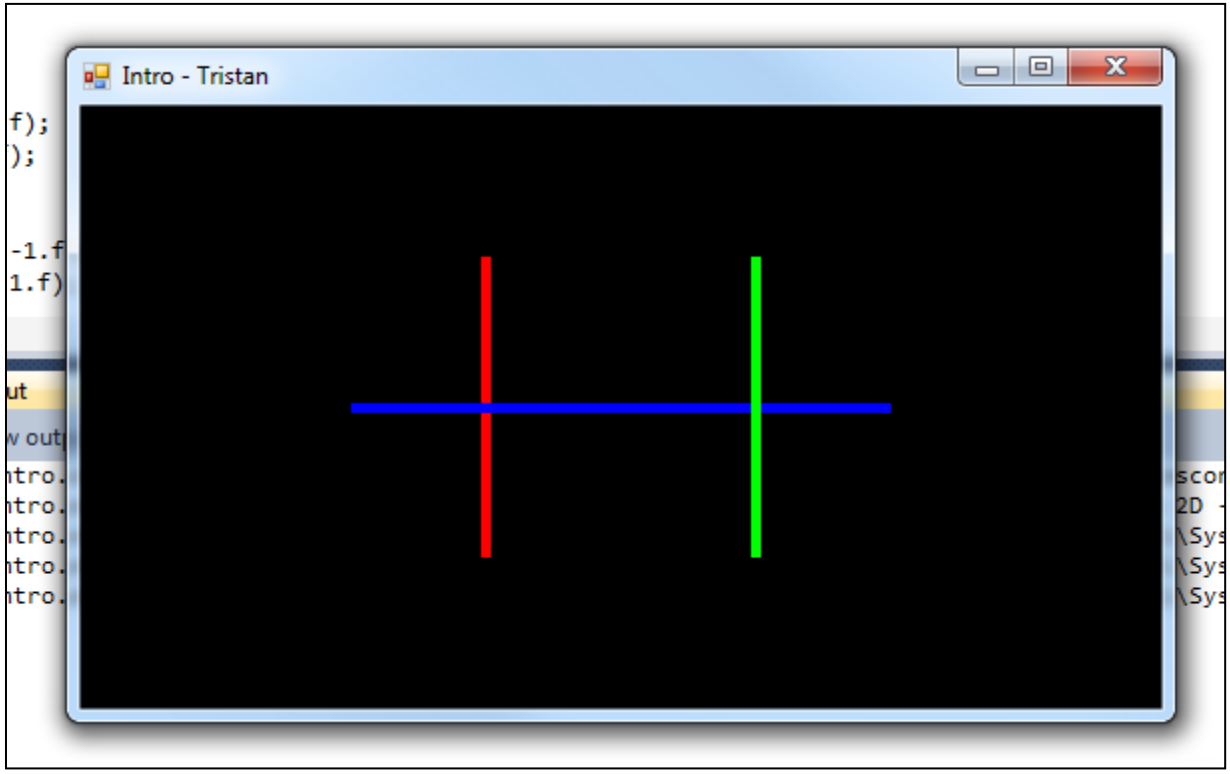
Enfin, le cœur du programme, le rendu :

```
133 System::Void Form1::Render(System::Void)
134 {
135     glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT); // Vide le tampon de couleur et de profondeur
136     glLoadIdentity();                                   // Réinitialise la matrice modèle/vue
137
138     glBegin(GL_LINES);                                  // Début - en mode ligbe
139
140     glColor3f(1.f,1.f,1.f);                             // Couleur des vertex (blanc)
141
142     glVertex3f( -1.f, 0.f, 0.f);                        // Point de départ
143     glVertex3f( 1.f, 0.f, 0.f);                         // Point d'arrivée
144
145     glEnd();                                            // ligne complète
146
147     SwapBuffers(m_hDC);                                 // on affiche le tout à l'écrant
148 }
```

Exercices

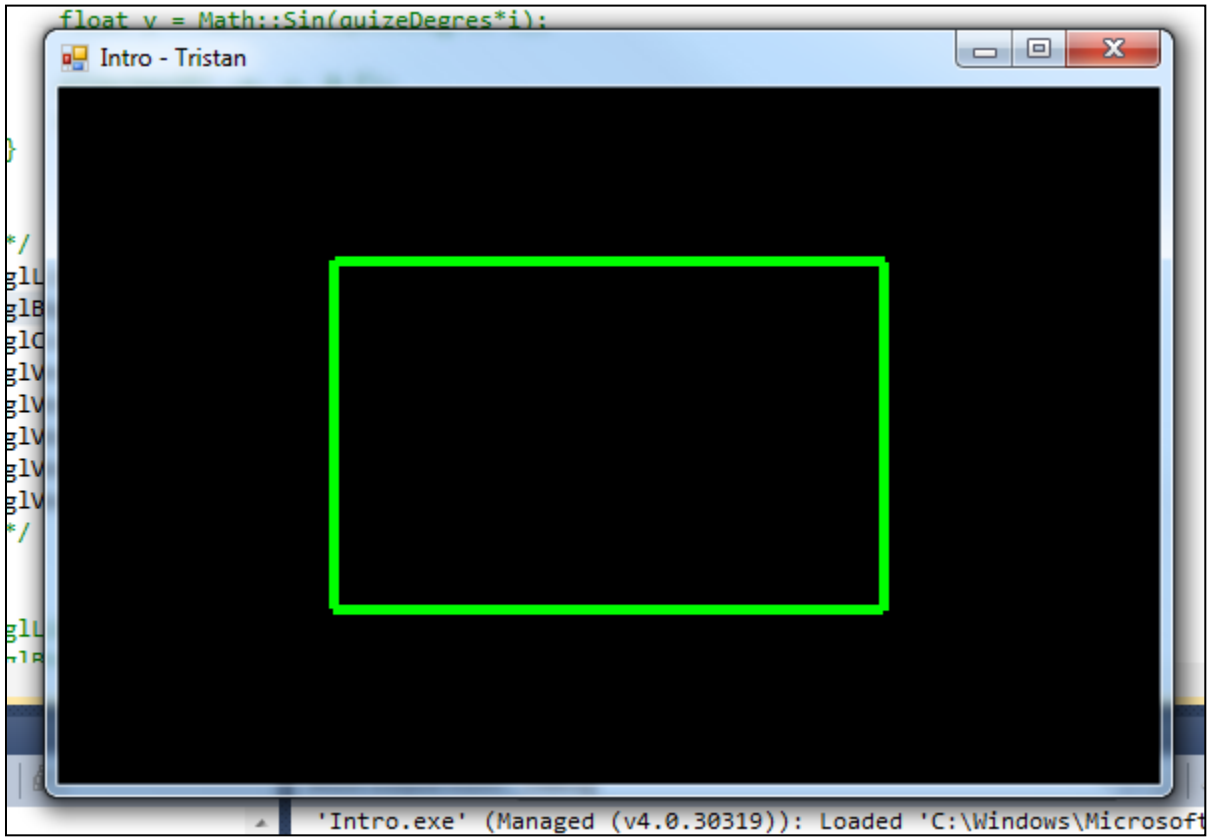
1. devant / derrière

À l'aide de le fonction « glLineWidth() », épaissir le trait et afficher une ligne devant et une ligne derrière la ligne originale. Utilisez des couleurs pour vous aider.



2. Un carré, pas vraiment carré

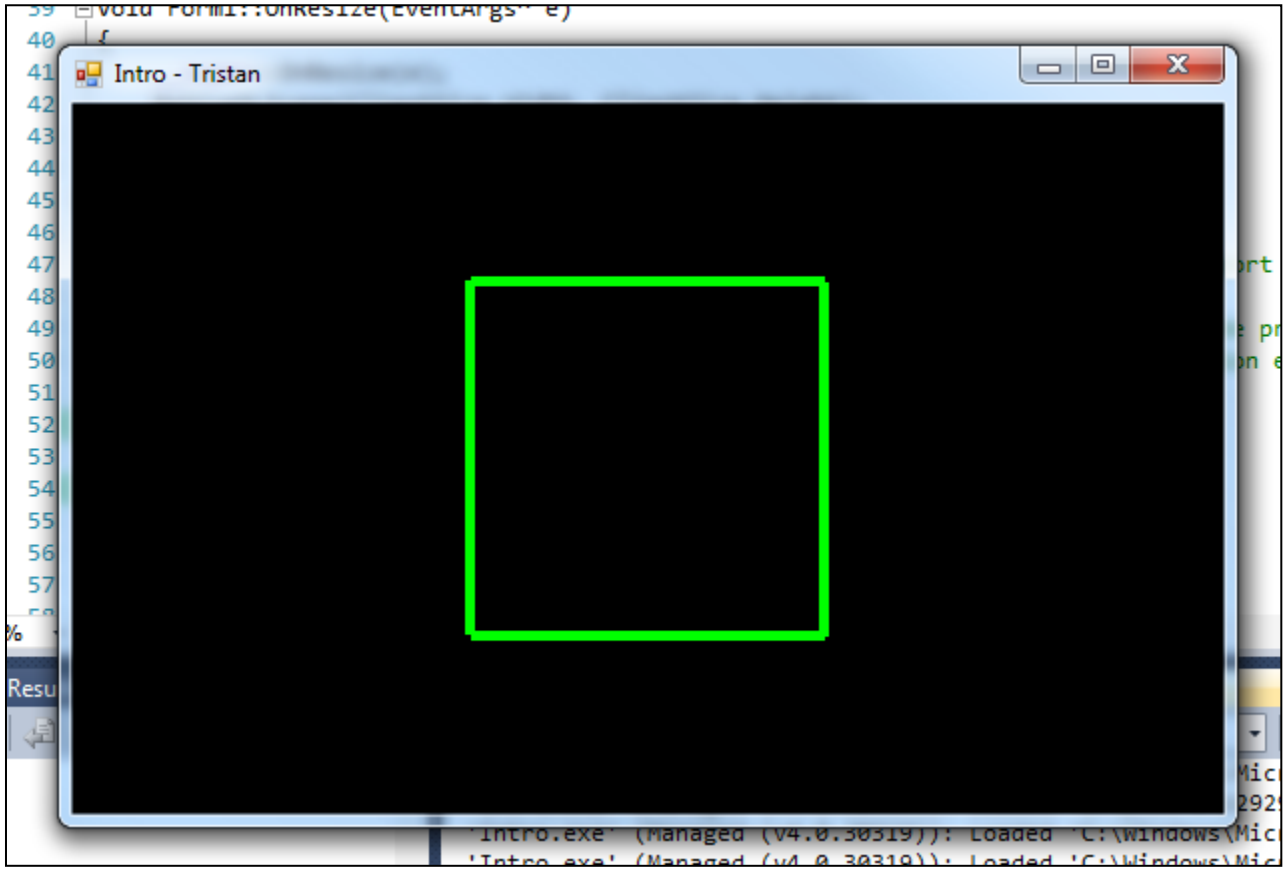
Cette fois nous allons utiliser « glBegin(GL_LINE_STRIP) » pour tracer un carré de dimension 2 X 2 (de -1 à +1 en x et y).



3. Vrai carré !

Ajustons notre matrice de projection pour avoir un carré.

(ca doit fonctionner que notre fenêtre soit plus large que haute ou le contraire)



4. Des maths…

Faisons appel à des opérations trigonométriques. Pour vous aider :

Math::Sin(), Math::Cos(), Math::PI