Open Inventor™ and MasterSuite

Patrick Barthelemy *G5G SA (a TGS company), Bordeaux, France*
Robert Weideman *TGS Inc. San Diego, USA*

Abstract

Open Inventor is a powerful C++ class library for building 3D graphics applications. Open Inventor was developed by Silicon Graphics Inc. for the UNIX/X11/Motif environment. The UNIX version consists of a portable core component and a window system (in this case X11 and Motif) specific component named “SoXt”. Template Graphics Software Inc. (TGS) has licensed the Open Inventor source and developed a version for Unix platforms and Microsoft Windows NT / Windows 95 platforms.

Keywords: TGS Inc., G5G SA, Open Inventor, MasterSuite, VRML, GraphMaster, 3D DataMaster

1 Architecture

Open Inventor was designed to be portable and window system independent. It uses a strategy similar to the one used in OpenGL (which Open Inventor normally uses as its rendering engine). OpenGL is divided into a large core of system independent functions and a small set of window (and operating system) specific functions. The system specific functions generally have similar functionality but different parameters. These OpenGL functions are identified by a unique prefix, for example:

UNIX X11	glX
Win32 (Windows NT/95)	wgl
OS/2	pgl
Macintosh	agl

This has been a successful strategy. Some modification of the program for each platform is necessary, but in most cases the window system specific calls are used in a small number of places. Of course there may be many other window system and user interface changes that must be made to the program, for example converting a Motif interface into the equivalent Win32 or MFC interface. These changes are independent of the graphics library.

The corresponding prefixes for Open Inventor are:

UNIX X11	SoXt
Win32 (Windows NT/95)	SoWin

The correspondence between Open Inventor components in the Win32 environment and in the UNIX environment is illustrated in Figure 1.

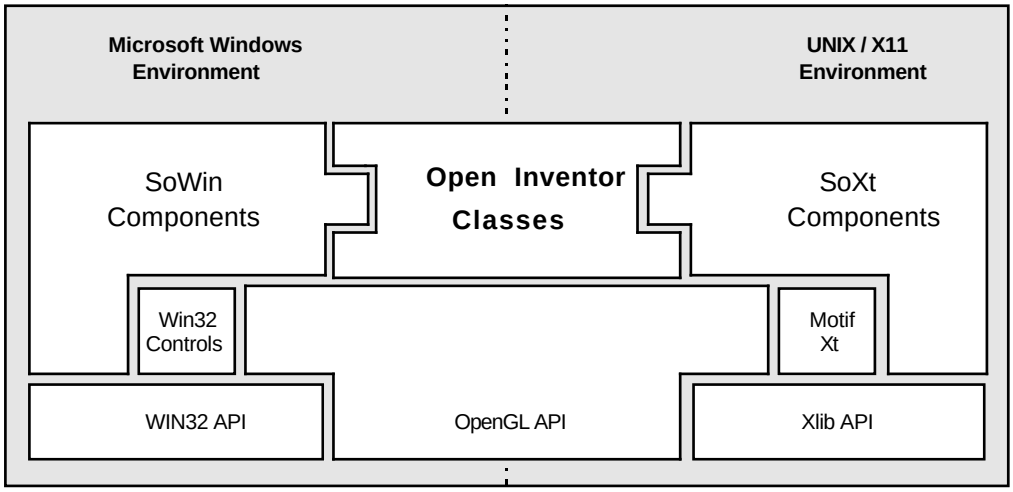

Figure 1: Open Inventor for Win32 and for UNIX

In porting an OpenGL program from UNIX you find that the Win32 "wgl" functions have different names and different syntax but similar concepts. We used the same strategy in designing the Open Inventor SoWin classes. SoWin contains the same classes as SoXt and in most cases their methods differ only in the data types of the parameters. This allows experienced UNIX Open Inventor programmers to benefit from their knowledge of the SoXt classes. It also allows all Open Inventor programmers to benefit from the extensive tutorial and example information in the *The Inventor Mentor* and *The Inventor Toolmaker*. SoWin is fully described in a following section.

There is a very close correspondence between the UNIX SoXt classes and the Win32 SoWin classes. This allowed us to define a set of SoXt "aliases" for the SoWin classes that facilitate porting Open Inventor programs from the UNIX X11 environment. Using the SoXt aliases, simple Open Inventor programs can often be ported to the Win32 environment with only minor changes. The SoXt aliases are described in their own section.

A significant difference between the Win32 C++ environment and UNIX is that much of the new C++ development under Win32 is being done using the Microsoft Foundation Classes (MFC) and the MFC AppWizard. MFC is both a class library that "wraps" the Win32 API and a powerful application framework based on the document/view paradigm. MFC provides a standard implementation of services common to many applications, such as printing, tool bars and status bars. The MFC AppWizard is a tool that generates a skeleton MFC application that is ready to compile, link and execute. The AppWizard can automatically add support for various features such as OLE, ODBC (database access), and so on.

For programmers using MFC, Open Inventor for Win32 provides the IVF classes and the IVF AppWizard. IVF is both a class library that "wraps" the SoWin API (similar to the way MFC wraps the Win32 API) and an extension of the MFC application framework specifically for 3D graphics applications. Using Open Inventor features, IVF also extends MFC to provide a standard implementation of 3D graphics services

common to many applications, such as interactive viewers, object manipulators and attribute editors. The IVF AppWizard is an extension of the MFC AppWizard that generates a skeleton MFC application with 3D graphics enabled. The IVF AppWizard can automatically add support for features such as interactive viewers, VRML "anchors" and so on. IVF is described in its own section.

So there are actually three parts to the implementation of the window-system-specific components of Open Inventor for Win32:

SoWin	Win32 specific classes
SoXt	Aliases for SoWin classes
IVF	MFC extension classes

Why three parts instead of just integrating Open Inventor into MFC? In short, because it provides the "best of both worlds". IVF provides a good, clean extension to MFC without having to expose every method of every Open Inventor component class. Similar to the way MFC allows direct access to the Win32 API, the Open Inventor component methods are accessible to applications that need them. At the same time the one-to-one correspondence between SoWin and SoXt classes allows Open Inventor programmers to carry over knowledge from one environment to the other and makes resources such as *The Inventor Mentor* directly applicable to the Win32 environment. The SoWin classes provide critical flexibility in two areas that will allow Open Inventor to be more quickly adopted and have lasting value in this environment. First, the SoWin classes can be used to build extension classes for application frameworks other than MFC. Second, the SoWin classes can be used to add 3D graphics functionality to existing applications, even in situations where development policy does not allow the class inheritance tree to be modified (as would be required to add IVF support to an MFC application).

2 SoXt Alias Classes

Here is a simple Open Inventor program from chapter 2 of *The Inventor Mentor* (example 2-4). The only difference between this program and the UNIX/X11 version shown in the Mentor is the addition of three lines (highlighted below). These lines use the standard preprocessor symbol "WIN32," which is defined in the Win32 environment, to change the name of function "main". Most Win32 programs (the exception is console applications) do not have a function named main. Instead the first entry point in the program is named "WinMain" and has a very different set of parameters. Applications using Open Inventor for Win32 are not required to have a WinMain because the optional SoWin utility library provides one. Open Inventor's WinMain function performs some basic bookkeeping operations and then calls the application's "ivMain" function with the same parameters as the familiar UNIX "main" function.

Example 1 “Hello Cone” Using the SoXt Alias Classes

```
#include <Inventor/Xt/SoXt.h>
#include <Inventor/Xt/viewers/SoXtExaminerViewer.h>
#include <Inventor/nodes/SoCone.h>
#include <Inventor/nodes/SoMaterial.h>
#include <Inventor/nodes/SoSeparator.h>
#if defined(WIN32)
#define main ivMain
#endif
void main(int, char **argv)
{
  Widget myWindow = SoXt::init(argv[0]);
  if (myWindow == NULL) exit(1);
  SoSeparator *root = new SoSeparator;
  root->ref();
  SoMaterial *myMaterial = new SoMaterial;
  myMaterial->diffuseColor.setValue(1.0, 0.0, 0.0);
  root->addChild(myMaterial);
  root->addChild(new SoCone);
  SoXtExaminerViewer *myViewer =
                    new SoXtExaminerViewer(myWindow);
  myViewer->setSceneGraph(root);
  myViewer->setTitle("Examiner Viewer");
  myViewer->show();
  SoXt::show(myWindow);
  SoXt::mainLoop();
}
```

Because the WIN32 symbol is not defined on UNIX systems, this program is completely portable. It can be compiled, linked and executed with no additional changes in either a Win32 or a UNIX environment. With one or two exceptions, all the example programs from *The Inventor Mentor* and *The Inventor Toolmaker* were made portable with only minor changes similar to the ones shown above. This is highly desirable because these programs are, for the most part, intended to illustrate system independent features of Open Inventor. Using the SoXt alias classes, the important information in the programs is not obscured by platform differences.

The SoXt alias classes are valuable for more than just simple example programs. Potentially any Open Inventor program that does not make explicit calls to Xlib, Xt or

Motif functions can be ported to Win32 with its SoXt calls intact. For example, the Open Inventor demo program “slotcar” (included with the Open Inventor for Win32 SDK) is a moderately complex program that was ported using the SoXt aliases. Even when there is Motif code to be converted, knowing that SoXt calls map directly to SoWin calls (and generally have the same behavior) means that the SoXt calls could be left in the code, which would reduce the number of “ifdefs” in the final code. We do not recommend this approach for production code but it can make porting easier for some projects.

Strictly speaking, there are no SoXt classes or data types in Open Inventor for Win32. However all the SoXt header files are provided. These header files redefine the SoXt classes as the corresponding SoWin classes and the necessary X11 (and Xt and Motif) data types as the corresponding Win32 data types. So, for example, class SoXtExaminerViewer is redefined as class SoWinExaminerViewer. There is an SoWin class that corresponds to each of the SoXt classes. The Win32 SoXt (SoWin) methods generally perform the same, or very similar, actions as their UNIX counterparts.

In order to redefine the X11 data types, we take advantage of the general similarity between the X Window System and the Microsoft Window System. Both refer to windows using opaque handles, both provide a window hierarchy in which child windows are clipped against their parent window, and so on. The X11 type *Window* corresponds well to the Win32 type *HWND* (which is the mapping Open Inventor uses). Win32 does not have a concept directly analogous to the Xt “widget”. However, in Xt there is always exactly one window that corresponds to a particular widget, so Open Inventor for Win32 also maps the Xt type *Widget* to the Win32 type *HWND*. Thus the SoXt::init method returns a “Widget” which is actually a window handle which can then be passed to the SoXtExaminerViewer constructor as its parent window.

The SoXt::show method has the same effect as its UNIX equivalent in this situation. It uses the Win32 function ShowWindow to make the application’s top level window visible. The SoXt::mainLoop method also has essentially the same behavior as its UNIX equivalent. It uses the SoXt::nextEvent and SoXt::dispatchEvent methods, which are in turn implemented using (mostly) the Win32 GetMessage and DispatchMessage functions. More details are given in the SoWin section. The Win32 *MSG* data type is roughly equivalent to the X11 *Xevent* data type. Most of the same basic user interface events exist in both window systems, for example keypress, mouse button and mouse motion events.

3 VRML and extensibility

IVF provides a convenient layer for implementing Win32 specific extensions to the core Open Inventor capabilities without having to modify Open Inventor itself (and thereby introduce portability problems). A good example is the standard VRML feature support defined for IVF. VRML stands for Virtual Reality Modelling Language. It is a standard file format, based on Open Inventor, for using 3D geometry and scenes on the World Wide Web. Open Inventor itself can read and write VRML geometry files (in addition to Open Inventor geometry files) and supports a superset of the nodes defined in VRML. However reading VRML nodes into the scene graph and actually providing browser-like behavior is two different things.

For example, the WWWAnchor node provides a way to associate a hyper-link with some geometry in the scene. Fields in the WWWAnchor node allow both a URL and some

descriptive text to be specified for the link. The WWWAnchor class allows the application to specify a callback function to be called when the Anchor's geometry is selected. So far, so good. But typically an application using WWWAnchor also wants to have some feedback provided to the user when the cursor is over the associated geometry, in other words when a click (or other action) would select the Anchor. IVF provides this behavior. By default the URL string is displayed in the mainframe window's status bar when the cursor is over the geometry. An application can override this behavior in the usual MFC way, by overriding a virtual function.

As was mentioned earlier, IVF implements support for VRML "hints", including ViewPoints. ViewPoints, for example, are specified as a set of predefined cameras under a Switch node named "Cameras". IVF puts the viewpoints (cameras) into a convenient list from which the application can set them. The IVF AppWizard can optionally create a ViewPoints menu for the application. If this menu exists, IVF will automatically add the viewpoints from the scene to the menu.

4 IVF AppWizard

The MFC AppWizard is a standard part of the Microsoft Visual C++ product. It is a powerful (and often used) tool that generates a skeleton MFC application that can include support for OLE, printing and many other features. The resulting application is ready to compile, link and execute. The IVF AppWizard is a "Custom AppWizard" for the Visual C++ 4.0 environment that extends the standard MFC AppWizard. In addition to the standard AppWizard dialog boxes, it displays the dialog box shown below. The IVF AppWizard generates an MFC application that includes support for interactive 3D graphics using IVF and the Open Inventor class library. Notice that the optional features include support for VRML nodes.

5 General Issues

The core Open Inventor classes (nodes, actions, sensors, etc) are almost entirely platform independent. However there are some cases where platform differences must be addressed, either by establishing a convention or by enhancing the classes. The most significant cases are:

5.1 Fonts

In the UNIX/X11 environment, Open Inventor's text classes can use any PostScript (Adobe Type1) font. A selection of PostScript fonts is available on every major UNIX/X11 platform. The default font in this environment is "Utopia", which is distributed with the Open Inventor SDK and assumed to be always available.

In the Win32 environment, PostScript fonts could be used, but they are not as commonly available because they are not distributed with the windowing system. Instead Open Inventor's text classes use Microsoft standard TrueType fonts. A basic set of these fonts is available on every Windows NT or Windows 95 platform, and many different fonts are available from third parties. The default font in this environment is "Times New Roman", which is distributed with the Windows NT and Windows 95 operating systems.

5.2 SoFont Node

In the UNIX/X11 environment, to use an alternate style of a PostScript font, for example bold or italic, you actually specify a different font. Usually the names of the variants are fairly obvious, for example “Utopia-Bold” and “Utopia-Italic”, but the point is that the UNIX/X11 environment uses the name of the font to encode style information in addition to the actual font name.

In the Win32 environment, each TrueType font (typically) supports some number of “styles”. The four basic styles are “Regular”, “Bold”, “Italic” and “Bold Italic”. To use an alternate style of a TrueType font, you actually specify the *same font* but with a different style. Therefore, the Win32 environment needs more than just the name of the font to determine the desired appearance.

Open Inventor for Win32 uses the following convention for a string specifying a font (specifically for the *name* field of the SoFont node):

“Font Name : Style”

For example, a bold serif font could be specified as: “Times New Roman : Bold”. Note that font names under Win32 can (and often do) contain spaces. Space characters immediately preceding or following the colon character are ignored (treated as white space). The TrueType fonts used for the various values of the *family* field of the SoFontStyle node are:

SERIF	Times New Roman
SANS	Arial
TYPEWRITER	Courier New

5.3 File Portability

Open Inventor “ascii” format files are inherently portable because they are simply “text” files. The only issue is the minor difference in how a “line” is terminated. In the UNIX environment, each line of a text file is terminated by a *new line* character, also known as the *line feed* character. In the Win32 environment, many programs still use the DOS convention that each line of a text file is terminated by a *carriage return* plus a *line feed*. Open Inventor for Win32 reads both kinds of text files, but always writes files with UNIX style line termination (*line feed* only). Programs to do the (simple) conversion between line termination styles are widely available.

Open Inventor “binary” files face the usual portability issues of *byte order* and *floating point format*. Open Inventor for Win32 follows these conventions in reading and writing binary files:

Byte order	Same as SGI (“big endian”)
Floating point format	IEEE floating point
Numeric values	32 bits

Necessary format conversions are done automatically when reading or writing a file. Because Open Inventor uses the same conventions on every platform, Open Inventor binary data files are *network transparent*. The same file can be read successfully on any platform without performing any explicit conversion. Because the Open Inventor binary format is also a very compact representation, the portability conventions make it ideal for transferring and storing 3D graphics data.

5.4 Extended Input and Output

The standard Open Inventor classes, SoInput and SoOutput, can read and write Inventor data from a buffer in memory or a file represented by a *stdio* file pointer ("FILE *"). Both of these mechanisms are also available in the Win32 environment, but there are several additional requirements.

In order to allow derivation of new classes from SoInput and SoOutput, these classes were modified to make (essentially) all of their methods virtual. This change does not affect Open Inventor applications.

5.5 Offscreen Rendering

The Open Inventor SoOffscreenRenderer class is a very powerful tool for generating bitmap images of an Open Inventor scene. The bitmap image can be written out to a file or reused within the application as an image or texture map. Arguably, SoOffscreenRenderer belongs in the Open Inventor component library because it needs to use window system specific calls to obtain a bitmap (pixmap in X11 terms) that OpenGL can render into. However the system specific details are hidden from the application.

The standard SoOffscreenRenderer can write files in SGI's ".rgb" format or in Encapsulated PostScript format. Open Inventor for Win32 adds a new method to this class which writes a file in the standard Win32 ".bmp" format:

```
SbBool writeToBMP(FILE *fp) const;
```

This is much more useful in the Win32 environment because many programs can import and display ".bmp" files, including Microsoft's Word, Excel and Powerpoint. The standard SoOffscreenRenderer also provides a method (getBuffer) that returns a pointer to the bitmap image in OpenGL (glReadPixels) format. Open Inventor for Win32 extends this concept with a new method that returns a pointer to the Win32 memory device context (DC) that contains the image:

```
HDC getDC() const;
```

The memory DC is roughly equivalent to the glXPixmap used in the UNIX/X11 implementation. Because of the generality of the Win32 graphics model, a Win32 program can "blt" the bitmap image directly from the memory DC into a display window context, a Windows Enhanced Metafile (".emf") context, or a printer device context. A printer device context hides the details of the actual printer, allowing Open Inventor for Win32 programs to support hardcopy onto any device (supported by Windows) with no additional effort. The getDC method is also used in MFC programs to implement support for the "print preview" feature.

6 MasterSuite

The complete object oriented product line for all your cross-platform and cross-network graphics developments Powered by Open Inventor. This part is designed to help you understand what are DialogMaster, PlotMaster, GraphMaster and 3D DataMaster class libraries, the four companions of Open Inventor parts of the Master Suite by TGS-G5G.

Master Suite class libraries implement extensions to the Open Inventor development environment. These extensions give you access to all new functionality that will speed up your development by including 2D drawing and charting capabilities, a set of input devices that makes design of user interface much more powerful and still cross platform, a set of classes to develop viewers for your scientific data. This is not only for your screen but also for your favorite printer on which you are able to output your graphics using vector quality rather than just getting a hardcopy image through Open Inventor.

Because the Master Suite product line is powered by Open Inventor (and so OpenGL), you are going to develop fully cross-platform graphics applications and you will be able to deliver your graphics through the network using the de facto standard VRML. Using the 3Space family products your Master Suite visualization may be included using ActiveX control in a Word document, an Excel report, a PowerPoint presentation or in e-mail, to share with others real 3D information that can be manipulated and not only a flat and static 2D representation of your results. You can also distribute your 3D graphics to the Web by including them in your Web page, then people are able to visualize and manipulate your results.

If you develop your application using Master Suite you have still access to all Open Inventor capabilities, you can mix Master Suite and Open Inventor nodes, you can use all of Open Inventor rendering facilities such as camera, lighting, texturing, and fog to render your visualization. Master Suite node’s fields can be connected to any Open Inventor engines to animate them for instance or sensors to change any of their values. The same way you are able to store your Open Inventor database in a metafile, you can store and retrieve your visualization to or from a metafile that can be a iv format or VRML format.

6.1 DialogMaster

The first component of the Master Suite products line, DialogMaster extends the Xt component of Open Inventor and provide you with an easy and powerful way to design cross-platform user interfaces for your Master Suite application. This component library is available for both UNIX and windows platforms. DialogMaster defines a dialog area that can include sliders, choice button or string input areas. Each of the devices can be triggered and then a callback function may be started for each of them. All Master Suite (or Open Inventor) parameters may be changed using these devices. Some powerful « widgets » are also provided within DialogMaster to very easily customize GraphMaster and 3DDataMaster nodes such as axis editors, contouring editors, or cross section editors. GraphMaster monitors provide you with a powerful built-in user interface to customize your vector PostScript, HPGL or CGM outputs for all your Open Inventor or Master Suite applications.

6.2 PlotMaster

The second component of the Master Suite product line, PlotMaster extends Open Inventor actions to allow your application to output a vectorized format of your database. At this time Open Inventor can only make a postcript bitmap output of your database. This action stores the image of your visualization using PostScript image output, which is like doing a hardcopy of your screen. With PlotMaster you are able to create vectorized output using CGM, HPGL or PostScript format. With CGM format you can the plug your date in a wide range of other software, using HPGL or PostScript format you can now produce high quality paper output. PlotMaster allows you to fully define configuration parameters by setting the action fields to fit your printer configuration.

6.3 GraphMaster

GraphMaster gives you access to a new set of 2D and 3D shape nodes :

2D nodes : Open Inventor only provides 3D shape nodes. With this first set of extension, you are able to design very easily and efficiently your 2D graphic applications. This includes simple 2D shapes, such as rectangles, circles or arrows, 2D charting nodes to draw a range of different 2D axis (linear logarithmic, angular, time...), curves or markers field, statistical nodes using pie or bar chart representation, error areas, etc.

3D drawing : The previous classes are extended in 3D such as 3D circles, 3D pie or bar charts, 3D curves or 3D axis. These nodes give you access to a wide range of new 3D functionality:

- cartesian axis 2D/3D
  - linear
  - logarithmic
  - generalized
- angular axis 2D/3D
- polar axis
  - linear
  - logarithmic
- time axis 2D/3D
- multi-axis system 2D/3D
  - XY axis
  - rectangle of XY axis
  - XYZ axis
  - rectangle of XY axis with Z elevation
- curve
  - linear
  - stair
  - smooth (cubic-spline)
  - double stairs
  - histogram
- raise points
- filling curve to a threshold
- 2D/3D text
  - vectorized fonts
  - PostScript fonts
- legends
  - items legend
  - values legend
- statistic representations
  - single histogram
  - multiple histogram
  - 2D pie chart
  - 3D pie chart
  - high-low-close
  - error curve
  - error points field
  - points field bars
- 2D3D generalized
  - parallelogram
  - circle, circle arc
  - arrow
  - labels field
  - valued markers field

6.4 3D DataMaster

3D DataMaster allows you to « mine » your data. The best way to understand, explain or distribute your scientific results is certainly to visualize them. *3D DataMaster* can

deal with whatever type of meshes you have, 2D or 3D meshes, Cartesian, spherical, cylindrical, triangle, hexahedral, structured or indexed and let you build from them vectors field, 2D or 3D contouring, cross section and of course a wide variety of legends to explain your visualization.

6.4.1 : Types of meshes :

2D "grid" mesh :	cartesian, parallel cartesian, polar.
2D indexed mesh (unstructured):	mesh of triangles, mesh of quadrangles, and mesh of any type of polygon. Each element of the mesh is defined by its nodes indices.
3D "grid" mesh :	cartesian, parallel cartesian, cylindrical,spherical.
3D indexed mesh (unstructured):	mesh of tetrahedrons, mesh of hexahedrons, and mesh of pentahedrons, pyramids, hexahedrons or tetrahedrons. Each element of the mesh is defined by its nodes indices.

6.4.2 : 2D mesh representations :

limit :	representation of the mesh limits
lines :	representation of each edge of the mesh
filled :	fill each element of the mesh
contouring :	representation of contouring lines
side :	2D-1/2 representation of the mesh limits, by using altitude values
vector :	representation of 2D vectors field
streamline :	representation of 2D streamlines

6.4.3 : 3D mesh representations :

skin :	representation of the mesh skin
cross-section :	representation of scalar data on a cross-section of the mesh (cut-plane)
level surface :	3D level surface of scalar data on the mesh.
cross-contour	intersection of a cross-section and the mesh skin
skeleton :	set of X, Y and Z cross-contours which form a wireframe skeleton of the mesh
vector :	representation of 3D vectors field
streamline :	representation of 3D streamlines
voxel :	volumic representation

6.4.4 : Coloring type attribute of 2D-3D mesh representation

It defines the method used to color the edges or facets of the representation:

inherited : each elements of the representation are drawn with the same color

average: the color used to draw an element depend of the average of its nodes values.

mapping: data mapping

contouring : each element of the representation is divided into small elements according the contouring values

7 Contact TGS Inc. or G5G SA

Template Graphics Software, Inc.
9920 Pacific Heights Blvd,
Suite 200
San Diego, CA 92121 - USA
Phone : 619 457 5359
Fax :619 452 2547
Email : info@tgs.com
WWW : http://www.tgs.com/

G5G SA, a TGS company
European Headquarters :
PA Kennedy I - BP 227
33708 Mérignac Cedex - FRANCE
Phone : +33 (0) 5 56 13 37 77
Fax : +33 (0) 5 56 13 02 10
Email : info@g5g.fr
WWW : http://www.tgs.com/