OpenGL functionality in GINO v8.0

GINO has contained OpenGL functionality since v5.0. Right from the beginning the functionality was built in to GINO rather than supplied as a third-party DLL and this provides the following benefits:

- All software supplied and maintained by one company
- Complete printed and on-line documentation in Fortran-90 (no need to buy OpenGL books written for the C programmer)
- Full integration with existing 3D graphics routines
- Full integration with GINOMENU offering widget facilities over and above f90gl
- 3D object routines includes spheres, cubes, cones, volumes, wedges, spline surfaces and bezier surfaces
- Common GINO interaction model under Windows avoiding the need for Windows API calls
- Single call for Segment hitting
- Automatic calculation of planar normals which are essential for flat shading
- Automatic calculation of averaged normals for smooth shading of built-in 3D objects
- Facets generated with one routine (compared to a minimum of 6)
- Automatic grouping of adjacent primitives
- Lighting, shading and animation of 3D surfaces available with GINOSURF

The OpenGL functionality is fully integrated into the GINO libraries through proprietary subroutine calls which interface to OpenGL functionality in the OpenGL DLLs. Using a proprietary interface allows GINO to develop at its own pace and also allows for the development of any new 3D interface such as Direct3D without forcing the users to re-write their applications.

Performance

Performance of OpenGL programs can be improved in many ways, including those described below:

- Use as few light sources as possible
- Store objects in segments
- In animated objects, only use the facet primitive - most cards are tuned to draw only triangular facets at high speed
- Use as few changes to material properties within an object as possible - i.e. group together facets with the same material
- Cull back facing facets if possible
- Switch off back surface lighting
- Check the Depth Buffer capabilities of the graphics card and set GINO to match it

If performance or other problems persist, make sure that you have the latest video driver from the manufacturer - don't rely on the one supplied with the board as it will almost always be out-of-date.

Animation

Problems with GINO animation can be overcome by checking the following:

- The first GINO requirement is to use one of the 3D device-drivers:
    - If using GINO under Windows, use the **gWogl()** family of drivers
    - If using GINO under Linux use the **gGlx()** family of drivers.
    - If your application is based on using GINOMENU then you need to use a graphics frame with **gmFrameType=GOPENGL**
- Objects that are to be animated should be stored in segments. GINO has always had segment storage facilities and OpenGL cards use segment-storage for retaining objects in memory, so this will keep the performance at the maximum level.
- Double-buffering should be used where possible to speed up drawing and frame switching:
    - Calling **gStartBatchUpdate()** at the beginning of a segment will stop any further drawing going to the visible display surface and will only be sent to the backing-store.
    - Calling **gEndBatchUpdate()** at the end of a segment will then copy the backing store to the display surface usually resulting in a much faster display time.
- For 2D and 3D animation, GINO's full transformation and matrix-handling functionality is used to create the animation sequence either by segment transformations or use of the viewing functions to alter object position, projection or eye point.
- GINOSURF by nature contains its own 3D viewing capabilities, however for maximum performance and flexibility, these can be turned off using **gsSetSurf3D()** so that all 3D data is directed through to GINO and GINO's OpenGL device-driver.
- The other use of **gsSetSurf3D()** is for GINOSURF to use GINO's facet drawing facilities instead of basic polygons. Once this has been done, GINO's lighting and shading can be applied to the surface to give a much better appearance to the drawing.
- Again, once the surface has been defined within a segment and with the above settings, it can then be rotated in real-time simply by looping through a sequence of calls to a GINO viewing routine such as **gViewRotate()**.

See below for a list of OpenGL features with their availability in the GINO library at the current version. (This list is subject to change in future releases of GINO)

Feature	*Availability*
Vertex Primitives	
Points	gDrawPixel routine
Lines	All 2D/3D lines buffered up to form line strips
Line Strips	Normal 2D/3D line drawing
Line Loop	No
Triangles	gDrawFacet routine with 3 vertices

Triangle Strips/Fans	No
Quads	gDrawFacet routine with 4 vertices
Quad Strips	No
Polygon	gDrawFacet routine with > 4 vertices
Edge Flags	No
Normals	Automatic or optional arg in gDrawFacet routine
Vertex colours	Optional argument in gDrawFacet routine
Texture coordinates	Optional argument in gDrawFacet routine
Primitive attributes	
Line width and stipple	Yes
Culling	Yes
Polygon mode – solid/hollow	Yes
Facet Offset	Yes
Pixel facilities	
Drawing	Yes
Copying	Yes
Reading	Yes
Storage and transfer modes	No
Pixel buffer control	No
Pixel zooming	Software emulation
Bitmaps	No
Lighting and Colouring	
8 independent lights	Yes
Ambient, directional, point	Yes
Material properties	Via table or direct setting
Colour matching	Yes
Model shading	Flat and smooth
Lighting control	One/Two sided lighting switch only
Fog	Yes
Texture mapping	
1D images	No
2D images	Yes
Filtering control	Yes
Environment mapping	Yes
Viewing and clipping	
3D viewport	Yes
Parallel and perspective	Yes through existing viewing routines
3D clipping	Yes
Additional clipping planes	No

Modelling	
Shift, scale, rotate	Through existing routines
Matrix control	Through existing routines
Display Lists	
Creation	Through existing segment facilities
Deletion	Through existing segment facilities
Referencing	Through existing segment facilities
Drawing buffer	Yes
Depth buffer	Yes
Stencil/Accumulation buffer	No
Functions and tests	
Blending	Yes
Alpha, Dithering, Scissor	No
Logical operations	No
Selection and Feedback	
Selection	Yes through single gEnqSegHit routine
Feedback, Hints, Evaluators	No
Inquiries	Proprietary routines only
OpenGL Utility Library	
Manipulating images	No
Matrix operations	Yes
Polygon Tessellation	No
High level objects	Spheres and cylinders plus proprietary objects
NURBS	No
Error handling	Proprietary routines only
Windows extensions	
Font handling	Yes
X Windows extensions	
Font handling	Yes