

# GROOP: Graphics using Object Oriented Programming

Reference Manual -- Version 0.3

March 5, 1993

Thomas J. Watson Research Center  
IBM Research Division  
P.O.Box 704  
Yorktown Heights, N.Y. 10598

KOVED at WATSON  
[koved@watson.ibm.com](mailto:koved@watson.ibm.com)

## Abstract

Traditional interactive 3-D graphics programming is often difficult and error prone, especially for non-graphics programmers. Even for those people who are experienced, it is a tedious, time-consuming task requiring considerable knowledge.

GROOP makes interactive 3-D computer graphics much easier for the inexperienced or novice graphics programmer and improves the productivity of the experienced programmer. These are achieved through the use of a high-level object-oriented programming library written in C++. The GROOP programming library provides a rich set of extensible building blocks upon which end-user applications can be constructed.

The GROOP programming library offers several advantages:

1. **Renderer Independence** - The programmer does not need to learn the myriad details of programming 3-D graphics API's. This is completely handled by GROOP.
2. **Object-Oriented Programming Interface** - Learning and using GROOP is simplified by using Object-Oriented technology. Instead of the plethora of similar function calls found in typical graphics API's, GROOP has a simpler structure and far fewer interfaces to learn because of its use of inheritance.
3. **High Performance** - GROOP maintains low overhead through caching and other data management strategies. This reduces the temptation for graphics programmers to want direct access to the graphics API's.

### GROOP Features

1. Geometric Objects
  - Points, Polylines, Polygons, Triangle Meshes (strips), Text
  - Geometric transformations
    - Scale, Translate, Rotate, General 4x4 matrix, Saving Matrices
  - Material property specification
    - Ambient, Specular, Diffuse, Emission, Shininess
2. Composite Objects
  - Compose complex objects from simple objects and/or other composite objects
  - Define geometric transformations to be applied to all objects in the composite
  - Objects contained in the composite may inherit material properties.
3. Cameras
  - Simple
  - Stereo
4. Lights
  - Infinite
  - Spot
  - Local

GROOP was motivated by necessity -- the need to quickly construct 3-D graphics for Virtual Worlds (Virtual Reality) applications. The challenge was to write a graphics software interface for C++ programmers, while maintaining fast rendering.

Part of the design of GROOP is to be renderer independent. The initial implementation of GROOP uses GL for rendering. However, future implementations will also support PHIGS (graPHIGS and/or PEX), and a renderer on PVS. Other renderers are certainly possible. The reason is that GROOP is carefully divided into two parts -- geometry/properties specification and rendering. Users of GROOP specify geometry/properties of 3-D objects. From these specifications, the renderers can draw the 3-D scenes. This allows greater portability of applications between platforms without having to rewrite the graphics application code.

## Preface

This reference manual is divided into sections that reflect the organization of the system as a whole:

- Vertex
- + • Color & Textures
- Windows (specifically, the GL-based version)
- Cameras
- Lights
- Geometric Objects
  - Material Properties
  - Geometric Transformations
  - Points
  - Lines
  - Triangle Meshes (Strips)
  - Polygons
  - Text
  - Composites of the above geometric objects
- An appendix containing examples that use the above mentioned geometric objects. The source code serves as examples on how to use GROOP for constructing 3-D geometric models.
  - Circle
  - Cone
  - Cylinder
  - CappedCone and CappedCylinder
  - Cube
  - Sphere
  - Torus
  - Ball
  - Axis

## Acknowledgments

The initial implementation of GROOP was a joint effort of members of the Virtual Worlds project. The initial design was started by Larry Koved. During the early phases of design and implementation, Wayne Wooten joined in and made invaluable contributions. Wayne has since returned to Georgia Institute of Technology. Jim Lipscomb has and Bob Wolfe have also contributed to the implementation, particularly in improving the camera models. Scott Robertson, also from Georgia Institute of Technology, made valuable contributions through his comments, suggestions, and construction of demos. Chris Codella provided insight and suggestions on how to get around C++'s quirks and profitably take advantage of some obscure features of the language. He has also read draft versions of the documentation and helped improve the clarity of the documentation and software. Management support was provided by J. Bryan Lewis. Numerous people have read and provided improvements to this document.

## Release Notes

| GROOP runs on graphics hardware with 24-bit Z-buffer, RGB mode and double buffering.

The GROOP package consists of two documents, the GROOP User's Guide and the GROOP Reference Manual (this documents). The code consists of a library (libgroop.a), source code for some of the GROOP classes, and sample applications. They can be found in the directory /usr/lpp/groop.

### Notable omissions in the current release

- Assignment functions for many of the classes
  - Splines and NURBS.
  - Mouse and keyboard interaction
  - Pick mode
  - PHIGS (PEX) support
- | • non-RGB mode (e.g., color map mode) for graphics adapters that do not support 24-bit Z-buffer,  
| RGB mode and double buffering.

If you are interested in writing the support for color map mode or other non-24-bit RGB mode hardware adapters, contact Larry Koved (KOVED at WATSON, koved@watson.ibm.com).

## Change History

### + Release 0.3 Change Notes

- + Texture maps were added for GRPolygon objects only. This does not apply to GRPolygonList objects or composites. If there is sufficient time for testing, texture maps will be supported on GRTriMesh objects.
- + Only RGB texture maps are supported in the initial release.
- + An RGB image of the GL frame buffer can be captured and:
  - + 1. Returned to the application
  - + 2. Written to a file in .rgb file format used by xpic (byte interleaved)
  - + 3. Written to a PostScript or Encapsulated PostScript file

### | Release 0.2 Change Notes

| A number of changes were made in the current release of GROOP. Most notably are the changes to | GRVertex, Cameras (GRCamera and GRStereoCamera) and the addition of new camera models, and new | transformation interfaces to GRGeometricObjects.

- | • GRVertex now contains a number of new functions to perform operations on vertices.
- | • The camera code has been completely changed.
- | • The implementation of GRLights has completely changed. As a result, some of the default behavior | of lights has changed.
- | • GRGeometricObjects now have simpler interfaces for basic transformations. However, the old | transformation functions are all available.

| Changes for version 0.2 of groop are denoted by a vertical bar (|) in the left or right margin of the text.

# Table of Contents

<b>1.0 class GRVertex</b> .....	<b>1</b>
1.1 constructor .....	1
1.2 Normalize() .....	2
1.3 NormalizedCrossProduct() .....	3
1.4 DotProduct() .....	4
1.5 - (subtraction) / + (addition) .....	5
1.6 += (increment) .....	6
1.7 * (scalar multiply) .....	7
<b>2.0 class GRrgb</b> .....	<b>8</b>
2.1 constructor .....	8
+ <b>3.0 class GRTextureMap</b> .....	<b>9</b>
+ 3.1 constructor .....	9
<b>4.0 Window / Display classes overview</b> .....	<b>11</b>
<b>5.0 class GLwindow</b> .....	<b>12</b>
5.1 constructor .....	12
5.2 GLResizeWindow() .....	14
5.3 SetBackgroundColor() .....	15
5.4 Add() / Delete() - Geometric Objects .....	16
5.5 Add() / Delete() - Cameras and Lights .....	17
5.6 Display() .....	18
5.7 ResetTransformsOnDisplay() .....	19
+ 5.8 GetRGBImage() .....	20
+ 5.9 GetRGBImageToFile() .....	21
+ 5.10 GetRGBImageToPS() .....	22
5.11 destructor .....	23
<b>6.0 class GRCamera</b> .....	<b>24</b>
6.1 constructor .....	24
6.2 Reset() .....	26
6.3 Location() / ScreenPosition() / ScreenNormal() / ScreenUp() / ScreenWidth() / ScreenHeight() .....	27
6.4 SetCamera() .....	28
6.5 NearClippingDistance() / FarClippingDistance() .....	29
6.6 GetCamera() .....	30
6.7 CameraChanged() .....	31
<b>7.0 class GRStereoCamera</b> .....	<b>32</b>
7.1 constructor / SetStereoCamera() / GetStereoCamera() .....	32
<b>8.0 class GRSimpleCamera</b> .....	<b>33</b>
8.1 constructor / Location() / LookAt() / Twist() .....	33
<b>9.0 class GRSimpleStereoCamera</b> .....	<b>34</b>
9.1 constructor / Location() / LookAt() / Twist() / EyeSeparation() / ParallaxConvergenceDepth() .....	34
<b>10.0 class GRSimpleFixedScreenCamera</b> .....	<b>36</b>
10.1 constructor / Location() / LookAt() / Twist() .....	36
<b>11.0 class GRSimpleStereoFixedScreenCamera</b> .....	<b>37</b>

11.1	constructor / Location() / LookAt() / Twist() / EyeSeparation() / ParallaxConvergenceDepth()	37
11.2	ScreenPosition() / ScreenNormal() / ScreenUp() / ScreenWidth() / ScreenHeight() / NearClippingDistance() / FarClippingDistance()	39
<b>12.0</b>	<b>class GRLight</b>	<b>40</b>
12.1	constructor	40
12.2	Reset()	41
12.3	Color() / Ambient()	42
12.4	Location() / SpotDirection()	43
12.5	SpotLightAngleExponent()	44
12.6	LocalViewer()	45
<b>13.0</b>	<b>class GRGeometricObject</b>	<b>46</b>
13.1	Rotate / Scale() / Translate() / GenericTrans()	46
13.2	SetRotate() / SetScale() / SetTranslate() / SetScaleRotateTranslate()	47
13.3	SaveTransforms() / RemoveTransforms() / ResetTransforms()	48
13.4	ResetTransformsOnDisplay() / GetResetTransformsOnDisplay()	49
13.5	ReplaceTransforms()	50
13.6	material	51
<b>14.0</b>	<b>class GRMaterialProperties</b>	<b>52</b>
14.1	Emission() / Ambient() / Diffuse() / Specular() / Shininess() / Reset() / TextureMap()	52
<b>15.0</b>	<b>classes GRPoint and GRPointList</b>	<b>54</b>
15.1	GRPoint	54
15.2	GRPointList	55
<b>16.0</b>	<b>classes GRLine and GRLineList</b>	<b>57</b>
16.1	GRLine	57
16.2	GRLineList	58
<b>17.0</b>	<b>classes GRTriMesh and GRTriMeshList</b>	<b>59</b>
17.1	GRTriMesh	59
17.2	GRTriMeshList	60
<b>18.0</b>	<b>classes GRPolygon and GRPolygonList</b>	<b>62</b>
18.1	GRPolygon	62
18.2	GRPolygonList	63
<b>19.0</b>	<b>class GRText</b>	<b>65</b>
19.1	GRText	65
<b>20.0</b>	<b>class GRComposite</b>	<b>66</b>
20.1	GRComposite	66
<b>21.0</b>	<b>file readers (GRPointsReader, GRLinesReader, GRTriMeshReader, GRPolygonReader, GRCompositeReader)</b>	<b>68</b>
21.1	GRSimplePointsReader	69
21.2	GRSimpleLinesReader	70
21.3	GRSimpleTriMeshReader	71
21.4	GRSimplePolygonReader	73
<b>Appendix A. classes GRCircle, GRCone, GRCylinder</b>		<b>75</b>
A.1	GRCircle	75
A.2	GRCone	76

A.3	GRCylinder	77
<b>Appendix B. classes GRCappedCone, GRCappedCylinder</b>		<b>78</b>
B.1	GRCappedCone	78
B.2	GRCappedCylinder	79
<b>Appendix C. class GRCube</b>		<b>80</b>
C.1	GRCube	80
<b>Appendix D. classes GRTorus, GRSphere, GRBall, GRAxis</b>		<b>81</b>
D.1	GRTorus	81
D.2	GRSphere	83
D.3	GRBall	84
D.4	GRAxis	85

## 1.0 class GRVertex

### 1.1 constructor

#### Purpose

Create an object containing three floating point values (x, y, z), representing a point in 3-D space.

#### Libraries

GROOP library (libgroop.a)

#### Syntax

```
#include <GRVertex.h>
GRVertex vertex;
GRVertex vertex(x, y, z);
GRVertex vertex(array);

float x, y, z;
float array 3";
```

#### Description

The GRVertex creates an object containing x, y and z values used to define a 3-D vertex.

#### Parameters

- When no values are passed to the constructor, then (x, y, z) are defined to be (0.0, 0.0, 0.0).
- Arguments (x, y, z) are given the obvious assignments.
- When array is passed to the constructor, assignments are made as follows:  
x = array 0"; y = array 1"; z = array 2";

#### Implementation Specifics

Member variables **x**, **y** and **z** are public, and are therefore directly accessible (e.g., vertex.x, vertex.y and vertex.z).

#### Related Information

See GRrgb.

## | **1.2 Normalize()**

### | **Purpose**

| Change the GRVertex object into a unit length vector (length = 1).

### | **Libraries**

| GROOP library (libgroop.a)

### | **Syntax**

```
| #include <GRVertex.h>  
| void      vertex.Normalize();  
  
| GRVertex      vertex;
```

### | **Description**

| The GRVertex is treated as a vector, starting at the origin, and is scaled to be a vector of length 1.0 ("normalized").

### | **Implementation Specifics**

| The GRVertex, vertex, should not be (0, 0, 0). If it is, then a divide by zero will occur, with unpredictable results.

## | **1.3 NormalizedCrossProduct()**

### | **Purpose**

| Compute the normalized cross product of two vectors.

### | **Libraries**

| GROOP library (libgroop.a)

### | **Syntax**

```
| #include <GRVertex.h>  
| GRVertex vertex.NormalizedCrossProduct(x);  
  
| GRVertex vertex;  
| GRVertex x;
```

### | **Description**

| The cross product of the two vectors (vertex and x) is computed, normalized (to be a unit length vector) and returned.

### | **Parameters**

| One parameter, a GRVertex object, is the second vector used in computing the cross product.

### | **Return Value**

| The GRVertex object returned is a unit vector, with one point at the origin (0, 0, 0) and the GRVertex object returned is the other point in the vector.

### | **Implementation Specifics**

| The cross product should not be (0, 0, 0). If it is, then a divide by zero will occur, with unpredictable results.

## | **1.4 DotProduct()**

### | **Purpose**

| Compute the dot product of two vectors.

### | **Libraries**

| GROOP library (libgroop.a)

### | **Syntax**

```
| #include <GRVertex.h>  
| GRVertex vertex.DotProduct(x);  
  
| GRVertex vertex;  
| GRVertex x;
```

### | **Description**

| The dot product of the two vectors (vertex and x) is computed and returned.

### | **Parameters**

| One parameter, a GRVertex object, is the second vector used in computing the dot product.

### | **Return Value**

| The GRVertex object returned is the dot product of the two vectors.

## | **1.5 - (subtraction) / + (addition)**

### | **Purpose**

| Compute the difference or sum of two vertices.

### | **Libraries**

| GROOP library (libgroop.a)

### | **Syntax**

```
| #include <GRVertex.h>  
| GRVertex    A - B;  
| GRVertex    A + B;  
  
| GRVertex    A;  
| GRVertex    B;
```

### | **Description**

| The difference the two vertices is computed and returned:

| (A.x - B.X, A.y - B.y, A.z - B.z)

| Or, sum of two vertices is computed and returned:

| (A.x + B.X, A.y + B.y, A.z + B.z)

### | **Return Value**

| A GRVertex representing the difference between or the sum of two vertices.

## | **1.6 += (increment)**

### | **Purpose**

| Add a value to a vertex.

### | **Libraries**

| GROOP library (libgroop.a)

### | **Syntax**

```
| #include <GRVertex.h>  
| GRVertex A += B;  
  
| GRVertex A;  
| GRVertex B;
```

### | **Description**

| Add the value of the second vertex to the first vertex:

| (A.x += B.X, A.y += B.y, A.z += B.z)

## | **1.7 \* (scalar multiply)**

### | **Purpose**

| Multiply elements of a vertex by a scalar value.

### | **Libraries**

| GROOP library (libgroop.a)

### | **Syntax**

```
| #include <GRVertex.h>  
| GRVertex A * B;  
  
| GRVertex A;  
| float B;
```

### | **Description**

| Multiply each element of the scalar and return a vertex.

| (A.x \* B, A.y \* B, A.z \* B)

## 2.0 class GRrgb

### 2.1 constructor

#### Purpose

Create an object containing three floating point values (r, g, b), representing the three primary colors used for defining material or lighting properties.

#### Libraries

GROOP library (libgroop.a)

#### Syntax

```
#include <GRrgb.h>
GRrgb  rgb;
GRrgb  rgb(r, g, b);
GRrgb  rgb(array);

float  r, g, b;
float  array 3";
```

#### Description

The GRrgb creates an object containing r, g and b values used to define colors. The r, g and b values are defined to be in the range [0.0, 1.0].

#### Parameters

- When no values are passed to the constructor, then (r, g, b) are defined to be (0.0, 0.0, 0.0).
- Arguments (r, g, b) are given the obvious assignments.
- When array is passed to the constructor, assignments are made as follows:

```
r = array 0"; g = array 1"; b = array 2";
```

#### Implementation Specifics

Member variables **r**, **g** and **b** are public, and are therefore directly accessible (e.g., rgb.r, rgb.g and rgb.b).

#### Related Information

See GRVertex.

## + 3.0 class GRTextureMap

### + 3.1 constructor

#### + Purpose

+ Describe a texture map and how it is to be applied to a 3D object surface.

+ **Note:** This is the first pass implementation. The interface may change in the future.

#### + Libraries

+ GROOP library (libgroop.a)

#### + Syntax

```
+ #include <GRTextureMap.h>
+ GRTextureMap texture(width, height, map, mapping, expand,
+                       shrink, filter);
+ void texture.Add(x, y);

+     int           width, height;
+     char*         map;
+     TextureType   mapping;
+     TextureExpandFilter expand;
+     int           shrink;
+     TextureColorFilter filter;
+     float        x, y;
```

#### + Description

+ A texture map is a 2 dimensional (r,g,b) pixel map. The map can be applied to a surface, and the pattern can be repeated one or more times across the surface. In addition, the surface can be translucent or opaque.

+ **Note:** The initial implementation on GLwindow only supports opaque mappings.

#### + Parameters

+ <b>width</b>	Number of pixels per row of the texture map.
+ <b>height</b>	Number of rows of pixels in the texture map.
+ <b>map</b>	Points to an array of unsigned char bytes of length (width*height*3). The array is byte packed, three bytes (r,g,b) per pixel. Pixels are packed into the array starting from the top left pixel, scanning left to right for each row of pixels.
+ <b>mapping</b>	<b>TextureOnce</b> Apply the texture to the surface only once, no repetitions. <b>TextureRepeat</b> Repeat the texture map pattern one or more times, depending on the (x,y) mapping values. See the GRTextureMap Add() function below.
+ <b>expand</b>	This describes how to expand the texture map if the map is smaller than the surface being textured. TextureExpandClosesPoint means that for each surface pixel, the closes texture point will be used to color the pixel. TextureExpandAverage means that the texture map values will be averaged vertically and horizontally to compute the surface pixel color.
+ <b>shrink</b>	This describes how to shrink a texture map if it is bigger than the surface being textured.
+ <b>0</b>	Use the closest texture point when computing a pixel color.
+ <b>1</b>	Average the texture map values vertically and horizontally when computing the pixel color
+ <b>2</b>	A faster version of '0', but uses more memory in the renderer.

- + **4,5** Faster versions of '1', but use more memory in the renderer.
- + **filter** To create an opaque texture map, use TextureColorOpaque.
- + TextureColorTranslucent creates translucent texture maps (not currently supported).
- + **x,y** Each vertex in a surface needs to be mapped into a point in the texture map.
- + By definition, regardless of the width or height, texture maps are logically mapped into the space (0,0) in the lower left corner and (1, 1) in the upper right corner.
- + When defining the texture map, the Add() function needs to be called with texture points that will be mapped one-to-one with the vertices defined for the surface to which it is being applied.
- + To create repetition patterns for the texture, specify TextureRepeat, and the points Add() to the texture map will have x or y values greater than 1.0. For example, if a rectangle is defined as a polygon with 4 vertices, and the pattern should be repeated 5 times in the x direction and 10 times in the y direction, then Add(0,0) is used for one corner, Add(5,10) for the diagonal corner, Add(5,0) and Add (0,10) for the other two corners.
- + It is also true that non-uniform mappings are allowed from the texture map space to the object surface pixels, thus generating skewed texture maps. Skewed maps can generate interesting effects.
- + **Implementation Specifics**
- + Texture maps are only supported on some of the Silicon Graphics systems.
- + See the sample application Texture.C for an example of how to build and use texture maps.
- + **Related Information**
- + See GRMaterialProperties.

## 4.0 Window / Display classes overview

Scenes are computer graphics constructs in which 3-D objects are placed, with lights and a camera. The objects can be simple items such as lines, points, and text, or more complex constructs such as triangles and polygons. Each of these objects contain material properties, such as ambient, specular, diffuse and emission colors, and reflectance properties (e.g., shininess). Lights are used to illuminate and can be positioned anywhere within the scene. Cameras define what is to be drawn on the computer screen -- almost like a real camera with film. The window on the computer screen is similar to the film in a real camera. In general, windows or displays contain a camera, one or more lights and geometric objects that compose a scene to be viewed.

In GROOP, a display class, called `GRDisplay`, is defined as an "abstract class", or template, of what all display (or window) classes need to define (see `GRDisplay.h` for details). `GRDisplay` itself does not contain any code. From `GRDisplay`, other classes can be derived as subclasses which implement the member functions. In the current release of GROOP, there is one subclass: `GLdisplay`, which creates windows and renders scenes using the GL graphics library (`libgl.a`). Other future implementations may include PHIGS (e.g., `grPHIGS` or `PEX`) and PostScript.

## 5.0 class GLwindow

### 5.1 constructor

#### Purpose

Create a window (under X Windows) that will contain the output from the GL renderer.

#### Libraries

GROOP library (libgroop.a)  
GL library (libgl.a)  
USL C++ Standard Components library (lib++.a)

#### Syntax

```
#include <GLwindow.h>
GLwindow    gwindow(WindowTitle)
GLwindow    gwindow(WindowTitle, x1, x2, y1, y2)

    char*      WindowTitle;
    unsigned int  x1, x2, y1, y2;
```

#### Description

The GLwindow constructor creates a GL window under X Windows. An initial (non-stereo) camera and a single light source in default locations are created; the Z-buffer is cleared; rendering is in perspective mode. If x1, x2, y1, y2 are used in the constructor, then initial window sizing and placement are performed. If not, the window will be placed in the lower left corner of the display, with a size of 500x400 pixels (width x height).

#### Parameters

<b>WindowTitle</b>	the text string that appears on the window border supplied by the X Windows window manager.
<b>(x1, y1)</b>	defines the location of the lower left corner of the window when it is created. The coordinate system defines (0, 0) as the <i>lower left</i> corner of the display. <b>Note:</b> This is in contrast to X Windows, where (0, 0) is the upper left corner of the display.
<b>x2</b>	width of the window in pixels.
<b>y2</b>	height of the window in pixels.

#### Implementation Specifics

- When using stereo cameras, the x1, x2, y1, y2 arguments are ignored, and the window is the size of the display.
- When the environment variable *FULLSCREEN* is set, x1, x2, y1, y2 arguments are ignored, and the window is the size of the display.
- Z-buffered 24-bit RGB mode is used with double buffering.
- The GL graphics library (libgl.a) is used for rendering. The USL C++ Standard Components library (lib++.a) is also needed (list management).
- Rendering is done in perspective mode.
- Orthographic rendering is not implemented.

- + • GRText objects are rendered as 2-D text. Scaling and rotations do not affect the text, just the location of where the text is to be rendered. See class GRText for further details. On Silicon Graphics systems, font names are ignored. Only the default font is supported.
- + • GRTriMesh objects can only have up to 256 vertex/normal pairs. Having more vertices in a GRTriMesh object will yield unpredictable results.
- + • GLwindow supports up to seven (7) lights. If more lights are added, they are ignored during the rendering of the scene.
- + • Texture mapping is supported on those systems that have the appropriate hardware and software. This is currently limited to a subset of the Silicon Graphics systems.

### **Related Information**

See GRCamera, GRLight, and the classes derived from GRGeometricObject.

## 5.2 GLResizeWindow()

### Purpose

Change the size and location of the graphics window on the display.

### Syntax

```
#include <GLwindow.h>
void gwindow.GLResizeWindow(x1, x2, y1, y2)

    GLwindow      gwindow;
    unsigned int  x1, x2, y1, y2;
```

### Description

After a window is created, it is possible to move and/or resize the graphics window by using this member function.

### Parameters

The parameters are specified in the same manner as the GLwindow constructor.

### Implementation Specifics

This function does not appear to work correctly for all graphics adapters. It is suggested that the window be deleted and a new one constructed with the right size specified in the constructor.

## 5.3 SetBackgroundColor()

### Purpose

Set the color of the window background.

### Syntax

```
#include <GLwindow.h>
void gwindow.SetBackgroundColor(color)

GLwindow    gwindow;
GRrgb       color;
```

### Description

After a window is created, it is possible to change the color of the window's background.

### Parameters

**color**                    an GRrgb specifying the (r, g, b) values for the background color.

### Implementation Specifics

GRrgb values are specified as three float's with values between 0.0 and 1.0. SetBackgroundColor() scales these three values into the range 0 to 256 and casts them to integers.

### Related Information

See class GRrgb.

## 5.4 Add() / Delete() - Geometric Objects

### Purpose

Add a geometric object to the scene to be rendered, or delete an object previously added (using the Add() member function) to the scene.

### Syntax

```
#include <GLwindow.h>
void gwindow.Add(object)
void gwindow.Delete(object)

GLwindow          gwindow;
GRGeometricObject* object;
```

### Description

Geometric objects can be added to the scene to be rendered in the window by using the Add() member function. All geometric objects in GROOP, such as GRLine, GRText, and GRTriMesh (among others) are defined as a subclass of GRGeometricObject. As such, they can be added to the scene through the use of Add().

Geometric objects previously added to a scene, through the use of the Add() member function, can be removed from the scene to be rendered in the window by using the Delete() member function.

### Parameters

**object** points to an object that is a subclass of GRGeometricObject.

### Implementation Specifics

- Objects added to the scene are not shown until the next time the Display() member function is called for the window.
- Generally, GRGeometricObject is not instantiated directly (it is a C++ an "abstract class"). Instead, subclasses are created from GRGeometricObject, and it is objects of these subclasses that are added to / deleted from the scene by using the Add()/Delete() member functions.
- Objects added to the scene through the Add() member function can be deleted from the scene with the Delete() member function. Objects added to a scene through means other than the Add() member function (e.g., objects part of GRComposite's) can not be deleted from the scene through the Delete() member function. Subobjects of composites must be deleted through Delete() calls on the composites themselves.

### Related Information

See the Display() member function for class GLwindow.

See classes GRGeometricObject, GRPoint, GRPointList, GRLine, GRLineList, GRTriMesh, GRTriMeshList, GRPolygon, GRPolygonList, GRText, and GRComposite.

## 5.5 Add() / Delete() - Cameras and Lights

### Purpose

Tell the window which camera or lights to use when drawing the 3-D scene.

### Syntax

```
#include <GLwindow.h>
void gwindow.Add(camera)
void gwindow.Add(stereoCamera)
void gwindow.Add(light)
void gwindow.Delete(camera)
void gwindow.Delete(stereoCamera)
void gwindow.Delete(const light)

GLwindow    gwindow;
GRCamera*   camera;
GRStereoCamera* stereoCamera;
GRLight*    light;
```

### Description

Each GLwindow can have one active camera (or stereo camera) and up to seven active lights. The Add() member function tells the window to either change the current camera or add a light to the scene. The Delete() member function tells the window to delete the current camera and use the default camera, or delete a light from the scene.

Each time a window is told to use a camera, it forgets about the previous camera. Therefore, the programmer needs to decide whether to delete the previous camera after the new camera is added to the scene or a camera is deleted.

Lights can be added to a scene. The Add() member function adds a new light to the scene, although only the first seven lights are used when rendering a scene.

### Parameters

<b>camera</b>	pointer to a GRCamera object or a subclass of GRCamera.
<b>stereoCamera</b>	pointer to a GRStereoCamera object or a subclass of GRStereoCamera.
<b>light</b>	pointer to a GRLight object.

### Implementation Specifics

- The lights and camera are used by the Display() member function. Therefore, the lights and camera must be passed to the gwindow object before the Display() member function is called to have any effect.
- It is up to the programmer to decide when to delete the camera and/or light objects.
- Any number of lights can be added to a scene. However, Display() only uses the first seven lights.
- Only one camera or stereo camera is permitted in a GL window at a time.
- If the camera is deleted, the default (non-stereo) camera is used.

### Related Information

See GRCamera, GRStereoCamera and GRLight classes.

## 5.6 Display()

### Purpose

Draw the 3-D scene described by the geometric objects added to the scene through the Add() member function, and using the camera and lights previously defined and added to the window with the Add() member function.

### Syntax

```
#include <GLwindow.h>
void gwindow.Display()

    GLwindow    gwindow;
```

### Description

The list of objects previously added to the scene is traversed and appropriate GL calls are performed to draw the scene using the current camera and lights.

### Implementation Specifics

After the 3-D objects are drawn, the front/back buffers are swapped, so the current scene is displayed. The Z-buffer is cleared and the background color is drawn into the buffer.

## 5.7 ResetTransformsOnDisplay()

### Purpose

Tells the Display() member function to either delete or save the list of transformations for all of the 3-D objects being rendered after they have been rendered.

### Syntax

```
#include <GLwindow.h>
int gwindow.ResetTransformsOnDisplay(boolean)

GLwindow gwindow;
int boolean; // either TRUE or FALSE
```

### Description

Each time the GLwindow is told to display the 3-D scene, the Display() member function can either leave the transformation list intact, or it can delete all of the transformations in the list. The advantage of deleting the transformations is evident when trying to animate a scene.

ResetTransformsOnDisplay(TRUE) tells the window to reset the list of transforms not previously saved by a SaveTransforms() calls on GRGeometricObjects Add()'d to the GLwindow.

ResetTransformsOnDisplay(FALSE) tells the window to not reset the list of transforms not previously saved by a SaveTransforms() member function call on specific GRGeometricObject's Add()'d to the GLwindow.

The ResetTransformsOnDisplay(FALSE) is used particularly when multiple windows are to draw the same 3-D scene, and the multiple windows share the same objects. For example, all of the windows call ResetTransformsOnDisplay(FALSE) before calling its Display() member function, except for the last window to be Display()'d. The last window uses the ResetTransformsOnDisplay(TRUE) to reset the transforms that define the animation of the objects in the scene.

### Return Value

the previous setting (TRUE / FALSE) of the function.

### Implementation Specifics

Deletion of transformations applies to all objects in a scene unless the object is specifically marked to not have its transforms reset on display (see GRGeometricObject for further details). In general, deletions of transforms can not be performed on an object-by-object basis by using the member functions described here. To reset or remove transforms on specific GRGeometricObjects, use ResetTransforms() and RemoveTransforms() member function calls on those objects.

### Related Information

The GRGeometricObject class has a number of member functions to manipulate, save and delete transformations. See the ResetTransforms() and RemoveTransforms() member functions for class GRGeometricObject to delete transformations for specific objects. Also, SaveTransforms() in GRGeometricObject to compute and save a list of transformations used, but not deleted by the Display() member function.

## + 5.8 GetRGBImage()

### + Purpose

+ Grab the current frame buffer (the currently displayed image) and return it to the caller.

### + Syntax

```
+ #include <GLwindow.h>
+ int gwindow.GetRGBImage(&width, &height, &buffer);

+     GLwindow  gwindow;
+     int       width;
+     int       height;
+     char*     buffer;
```

### + Description

+ It is possible to get a binary representation of the current display (frame buffer). The buffer returned is a pixel-by-pixel image of the frame buffer, with three bytes per pixel (red, green, blue). The buffer will be an array of length (width \* height) \* 3 bytes.

### + Parameters

+ The width and height are set by the subroutine. The values returned are in pixels.

+ The GetRGBImage() function allocates memory for the frame buffer image, and returns a pointer to the buffer. The buffer is byte interleaved (red, green, blue) (one byte per color), starting from the top left corner of the window. The data is logically organized as an array, buffer row“ column“ color“, where color is 0, 2“, column goes from 0 to width, and row goes from 0 to height.

+ If the buffer can not be read for any reason, height and width will be zero, and buffer will point to NULL.

### + Implementation Specifics

+ Since the frame buffer is double buffered, the front buffer (the one current visible) is read. The current implementation does not read the Z-buffer.

### + Related Information

+ See GetRGBImageToFile() and GetRGBImageToPS().

## + **5.9 GetRGBImageToFile()**

### + **Purpose**

+ Grab the current frame buffer (the currently displayed image) and write it to a file.

### + **Syntax**

```
+ #include <GLwindow.h>
+ int gwindow.GetRGBImageToFile(&width, &height, fileName);

+     GLwindow    gwindow;
+     int         width;
+     int         height;
+     char*      fileName;
```

### + **Description**

+ It is possible to get a binary representation of the current display (frame buffer) and write it to a file. The buffer returned is a pixel-by-pixel image of the frame buffer, with three bytes per pixel (red, green, blue). The buffer will be an array of length (width \* height) \* 3 bytes.

### + **Parameters**

+ The width and height are set by the subroutine. The values returned are in pixels.

+ fileName is the name of a file that is receive the contents of the frame buffer. The data is byte interleaved (red, green, blue) (one byte per color), starting from the top left corner of the window. The data is logically organized as an array, file row“ column“ color“, where color is 0, 2“, column goes from 0 to width, and row goes from 0 to height.

+ If the frame buffer can not be read for any reason, the file can not be opened for writing, then height and width will be zero.

### + **Implementation Specifics**

+ Since the frame buffer is double buffered, the front buffer (the one current visible) is read. The current implementation does not read the Z-buffer.

+ The file will be opened for writing. If it does not exist, then it will be created.

### + **Related Information**

+ See GetRGBImage() and GetRGBImageToPS().

## + 5.10 GetRGBImageToPS()

### + Purpose

+ Grab the current frame buffer (the currently displayed image) and generate a PostScript output file.

### + Syntax

```
+ #include <GLwindow.h>
+ int gwindow.GetRGBImageToPS(fileType, orientation, widthInInches,
+                             fileName, comments);

+     GLwindow  gwindow;
+     PSFileType fileType;
+     PSOrientation orientation;
+     float      widthInInches;
+     char*      fileName;
+     char*      comments;
```

### + Description

+ It is possible to get a PostScript file of the current display (frame buffer). The file can be in PostScript or Encapsulated PostScript format. The image can be drawn in portrait or landscape (rotate 90 degrees).

### + Parameters

+ **fileType** The output file can be PostScript (PSFileTypePS) or Encapsulated PostScript (PSFileTypeEPS) (see rgb2ps.h).

+ **orientation** The image can be drawn in a portrait orientation (PSorientPortrait), or can be rotated 90 degrees into landscape orientation (PSorientLandscape) (see rgb2ps.h).

+ **widthInInches** The image can be scaled. This is the width of the image (when looking at the page in a portrait orientation), regardless of its orientation (PSorientLandscape or PSorientPortrait).

+ **fileName** The name of the file to which the (Encapsulated) PostScript should be written.

+ **comments** A null-terminated string of comments about the image. The string should not contain non-printable or new-line characters. This comment will be included in the output file. This argument can be NULL if no comment is to be included in the file.

### + Return Value

+ If the file is not created for any reason (buffer could not be read or output file could not be created), then the return value will be 0 (FALSE). If it succeeds, the return value will be non-zero.

### + Implementation Specifics

+ Since the frame buffer is double buffered, the front buffer (the one currently visible) is read.

+ The file will be opened for writing. If it does not exist, then it will be created.

### + Related Information

+ See GetRGBImage() and GetRGBImageToFile().

## 5.11 destructor

### Purpose

Delete the 3-D graphics window.

### Syntax

```
#include <GLwindow.h>
delete gwindowPtr

GLwindow gwindow;
```

### Description

The class destructor cleans up some data structures and deletes the GL window.

### Implementation Specifics

The camera, lights and 3-D objects (GRGeometricObjects) in the scene are *not* deleted when the window object is deleted. Objects no longer needed by the application must be handled (e.g., deleted) by the application.

## 6.0 class GRCamera

### 6.1 constructor

#### Purpose

Class constructor. Create a camera (non-stereo) for use in rendering a scene.

#### Libraries

GROOP library (libgroop.a)

Math library (libm.a)

#### Syntax

```
#include <GRCamera.h>
GRCamera   gcamera;
GRCamera   gcamera(screenCenter, screenNormal,
                    screenUp, cameraLocation,
                    screenWidth, screenHeight,
                    nearClippingDistance, farClippingDistance);

GRVertex   screenCenter;
GRVertex   screenNormal;
GRVertex   screenUp;
GRVertex   cameraLocation;
float      screenWidth;
float      screenHeight;
float      nearClippingDistance;
float      farClippingDistance;
```

#### Description

GRCamera is usually subclassed to create cameras with simpler interface functions (c.f. GRSimpleCamera and GRSimpleFixedScreenCamera).

Creates a non-stereo camera that can be passed to a window as part of a renderable scene or passed to a GRStereoCamera. The graphics camera contains two components: (1) The rectangular screen upon which the 3-D images are projected,<sup>1</sup> and (2) the location of the camera (the observer).

The screen has a center, a width, height, a direction it is facing (it's normal vector) and an orientation or rotation (it's "up" vector).

The camera's location is the location of the observer when looking towards the screen.

The graphics camera is the screen/camera combination, and can be pictured as a *viewing frustum*, a four-sided pyramid. The rectangular base is the far clipping plane, and the camera is the tip of the pyramid where the four sides of the pyramid converge to a point.

The drawn image will appear to skew if the camera location is changed with respect to the screen center and screen normal. By changing the up vector, the image will appear to rotate. Changing the screen width or height will adjust the apparent scaling and aspect ratio of the image. Near and far clipping determine the distance from the camera where the rendering is clipped.

#### Parameters

**screenCenter**      The center of the screen.

**screenNormal**     A vector (whose origin is at (0, 0, 0)) which is perpendicular to the screen plane. The near and far clipping planes are also perpendicular to this vector.

<sup>1</sup> This is almost the equivalent of the film in a real camera.

	<b>screenUp</b>	A vector (whose origin is at (0, 0, 0)) that points toward the top of the screen.
	<b>cameraLocation</b>	The point where the camera (observer) is located.
	<b>screenWidth</b>	Screen width.
	<b>screenHeight</b>	Screen height.
	<b>nearClippingDistance</b>	Distance from the camera to the near clipping plane.
	<b>farClippingDistance</b>	Distance from the camera to the far clipping plane.
	<b>Implementation Specifics</b>	
	The default values for a GRCamera are defined by the Reset() function described below.	
	This class is typically used by classes derived from it rather than being used directly.	
	<b>Related Information</b>	
	See classes GLWindow, GLDisplay, GRStereoCamera, GRSimpleCamera and GRSimpleFixedScreenCamera.	

## 6.2 Reset()

### Purpose

Reset the camera to default values.

### Syntax

```
#include <GRCamera.h>
void camera.Reset()

    GRCamera    camera;
```

### Description

| Reset a variety of camera parameters: its location becomes (0, 0, 8); the screen is centered at the  
| origin; a normal along the z axis; it's up vector is along the y axis; screen width and height of 6;  
| near clipping at 0.1 and far clipping at 12000.0. See GRdefaults.h for details.

### Implementation Specifics

| Reset() calls Location(), ScreenPosition(), ScreenNormal(), ScreenUp(), ScreenWidth(),  
| ScreenHeight(), NearClippingDistance() and FarClippingDistance() to set the default values spec-  
| ified in GRdefaults.h.

### Related Information

| See Location(), ScreenPosition(), ScreenNormal(), ScreenUp(), ScreenWidth(), ScreenHeight(),  
| NearClippingDistance(), FarClippingDistance() and GRdefaults.h.

## | **6.3 Location() / ScreenPosition() / ScreenNormal() / ScreenUp() / ScreenWidth() / ScreenHeight()**

### | **Purpose**

| Set the camera location and the screen parameters to create the viewing frustum.

### | **Syntax**

```
| #include <GRCamera.h>
| void camera.Location(position)
| void camera.ScreenPosition(center)
| void camera.ScreenNormal(normal)
| void camera.ScreenUp(up)
| void camera.ScreenWidth(width)
| void camera.ScreenHeight(height)
|
| GRCamera camera;
| GRVertex position, center, normal, up;
| float width, height;
```

### | **Description**

| **Location()** the camera (the observer) is moved to a new location.

| **ScreenPosition()** the center of the screen.

| **ScreenNormal()** a vector perpendicular to the screen.

| **ScreenUp()** a vector the points toward the top of the screen.

| **ScreenWidth()** width of the screen.

| **ScreenHeight()** height of the screen.

### | **Parameters**

| The parameters are the same as those used in the Reset() function and the camera constructor.

### | **Related Information**

| See the GRCamera constructor, Reset() function, and GRStereoCamera. Default values are defined in GRdefaults.h.

## 6.4 SetCamera()

### Purpose

Set the camera's screen parameters and camera location.

### Syntax

```
#include <GRCamera.h>
void camera.SetCamera(center, normal, up, position,
                     width, height, near, far);
```

```
    GRCamera camera;
    GRVertex center;
    GRVertex normal;
    GRVertex up;
    GRVertex position;
    float width;
    float height;
    float near;
    float far;
```

### Description

A single call to set the screen location, normal, up vector, width, height; camera location; near and far clipping plane distances from the camera. See the class constructor and Reset() functions for further details.

### Related Information

See the class constructor, Location(), ScreenPosition(), ScreenNormal(), ScreenUp(), ScreenWidth(), ScreenHeight(), NearClippingDistance(), FarClippingDistance() and GRdefaults.h. Also, see GRStereoCamera.

## | **6.5 NearClippingDistance() / FarClippingDistance()**

### | **Purpose**

| Set the distance for the near and clipping planes.

### | **Syntax**

```
| #include <GRCamera.h>  
| void camera.NearClippingDistance(near)  
| void camera.FarClippingDistance(far)
```

```
| GRCamera camera;  
| float near;  
| float far;
```

### | **Description**

- | • NearClippingDistance() changes the distance from the camera location to the near clipping plane.
- | • FarClippingDistance() changes the distance from the camera location to the far clipping plane.

### | **Related Information**

| See the class constructor, Reset(), SetCamera() and GRdefaults.h.

## 6.6 GetCamera()

### Purpose

Obtain current camera parameters.

### Syntax

```
#include <GRCamera.h>
void camera.GetCamera(&center, &normal, &up, &right, &location,
                     &width, &height, &near, &far)
```

```
    GRCamera    camera;
    GRVertex*   center;
    GRVertex*   normal;
    GRVertex*   up;
    GRVertex*   right;
    GRVertex*   location;
    float       width;
    float       height;
    float       near;
    float       far;
```

### Description

The address of a buffer for each of the arguments is passed to the GetCamera() member function, which copies pointers to the the current values or the current values into the caller's buffers.

### Parameters

<b>center</b>	location of the center of the screen.
<b>normal</b>	a vector that is normal to the screen.
<b>up</b>	a vector that points toward the top of the screen.
<b>right</b>	a vector that points toward the right edge of the screen.
<b>location</b>	location of the camera (the observer).
<b>width</b>	width of the screen.
<b>height</b>	height of the screen.
<b>near</b>	distance from the camera to the near clipping plane.
<b>far</b>	distance from the camera to the far clipping plane.

## | **6.7 CameraChanged()**

### | **Purpose**

| Set the changed flag and return the previous setting.

### | **Syntax**

```
| #include <GRCamera.h>  
| int camera.CameraChanged(int boolean)  
| void camera.FarClippingDistance(far)
```

```
| GRCamera camera;  
| int boolean;
```

### | **Description**

| The changed flag is used by displays (e.g., GLwindow) when caching camera information. Every time one of the camera's parameters are set (e.g., Location(), ScreenPosition(), etc.), CameraChanged(TRUE) is called to set the internal changed flag. Displays, such as GLwindow, reset the flag (CameraChanged(FALSE) after the camera's new values have been read and new cache values computed.

### | **Parameters**

| boolean is either TRUE or FALSE.

### | **Return Value**

| The value returned is TRUE or FALSE, and represents the prior setting of the changed flag.

### | **Related Information**

| See the class constructor, Reset(), SetCamera() and GRdefaults.h.

## 7.0 class GRStereoCamera

### 7.1 constructor / SetStereoCamera() / GetStereoCamera()

#### Purpose

Class constructor. Create a stereo camera for use in rendering a scene stereoscopically (two images of the same scene).

#### Libraries

GROOP library (libgroop.a)

#### Syntax

```
#include <GRStereoCamera.h>
GRStereoCamera    gcamera;
GRStereoCamera    gcamera(leftCamera, rightCamera);
void    gcamera.SetStereoCamera(leftCamera, rightCamera);
void    gcamera.GetStereoCamera(&leftCamera, &rightCamera);

    GRCamera*    leftCamera;
    GRCamera*    rightCamera;
```

#### Description

GRStereoCamera is usually subclassed to create cameras with simpler interface functions (c.f. GRSimpleStereoCamera and GRSimpleStereoFixedScreenCamera).

Creates a stereo camera that can be passed to a window as part of a renderable scene. The stereo camera contains two regular cameras (GRCamera objects) that can be manipulated independently.

A default stereo camera does not contain any cameras. In this case, a window will use default cameras.

Cameras can be replaced by using the SetStereoCamera() function.

GetStereoCamera() returns pointers to the current cameras.

#### Parameters

**leftCamera**            the camera for the left eye.

**rightCamera**           the camera for the right eye.

#### Implementation Specifics

SetStereoCamera() does not delete the old cameras.

The destructor for GRStereoCamera will delete the current cameras.

#### Related Information

See classes GLwindow, GLDisplay, and GRCamera, GRSimpleStereoCamera and GRSimpleStereoFixedScreenCamera.

## | 8.0 class GRSimpleCamera

### | 8.1 constructor / Location() / LookAt() / Twist()

#### | Purpose

| A simplified camera is created. It behaves like a camcorder. You can move the camera around, changing its location and orientation, and it will automatically update the underlying GRCamera.

#### | Libraries

| GROOP library (libgroop.a)

#### | Syntax

```
| #include <GRSimpleCamera.h>
| GRSimpleCamera gcamera;
| void gcamera.Location(location);
| void gcamera.LookAt(direction)
| void gcamera.Twist(rotation)
|
| GRVertex location;
| GRVertex gcamera;
| float rotation;
```

#### | Description

| GRSimpleCamera is a simplified GRCamera that acts like a camcorder. You can move the camera's location and set the direction that it is facing, as well as rotating it (giving a twist).

#### | Parameters

<b>location</b>	change the camera (observer's) location. The camera still faces the previous direction point. The default is to be at (0, 0, 8).
<b>direction</b>	the camera is turned to face towards this new point. The location of the camera does not change. The default direction is (0, 0, 0).
<b>rotation</b>	counter clockwise rotation of the camera. The rotation is along the line specified by the camera location and direction points.

#### | Implementation Specifics

| Default location and direction values are taken from GRCamera. The rotation is computed from the default GRCamera values.

| Changing the location does not change the direction or rotation values. Similarly, changing direction does not change location or rotation, and changing rotation does not change location or direction.

#### | Related Information

| See classes GLwindow, GLDisplay, and GRCamera, GRStereoCamera, GRSimpleStereoCamera and GRSimpleStereoFixedScreenCamera.

## 9.0 class GRSimpleStereoCamera

### 9.1 constructor / Location() / LookAt() / Twist() / EyeSeparation() / ParallaxConvergenceDepth()

#### Purpose

A simplified stereoscopic camera is created. It behaves like a camcorder. You can move the camera around, changing its location and orientation, and it will automatically update the underlying GRCameras.

#### Libraries

GROOP library (libgroop.a)

#### Syntax

```
#include <GRSimpleStereoCamera.h>
GRSimpleStereoCamera  gcamera;
void  gcamera.Location(location);
void  gcamera.LookAt(direction)
void  gcamera.Twist(rotation)
void  gcamera.EyeSeparation(width)
void  gcamera.ParallaxConvergenceDepth(depth)
```

```
GRVertex  location;
GRVertex  direction;
float     rotation;
float     width;
float     depth;
```

#### Description

GRSimpleStereoCamera is a simplified GRStereoCamera that acts like a camcorder that produces stereoscopic images (assuming you have the necessary stereoscopic display hardware). You can move the camera's location and set the direction that it is facing, as well as rotating it (giving a twist). The distance between the cameras can be set, as well as the distance to the plane where the images for the two eye are coplanar (zero parallax).

#### Parameters

<b>location</b>	change the camera (observer's) location. The camera still faces the previous direction point. The default is to be at (0, 0, 8).
<b>direction</b>	the camera is turned to face towards this new point. The location of the camera does not change. The default direction is (0, 0, 0).
<b>rotation</b>	counter clockwise rotation of the camera. The rotation is along the line specified by the camera location and direction points.
<b>width</b>	the distance between the two cameras.
<b>depth</b>	the distance from the midpoint between the two eyes to the cameras' screen. The screen is the plane of zero parallax. This means that objects behind the plane will appear to be behind the screen of the real display. Objects between the screen and the camera will appear to be in front of the real display.

#### Implementation Specifics

| The simple stereo camera consists of two GRCameras, each separated by the EyeSeparation width.  
| All computations for location, direction, rotation and parallax convergence depth (distance to the  
| camera's screen) are computed from the midpoint between the two cameras.

| Default location and direction values are taken from GRCamera. The rotation is computed from  
| the default GRCamera values.

| Changing the location does not change the direction or rotation values. Similarly, changing di-  
| rection does not change location or rotation, and changing rotation does not change location or  
| direction.

| **Related Information**

| See classes GLWindow, GLDisplay, and GRCamera, GRStereoCamera, GRSimpleCamera and  
| GRSimpleStereoFixedScreenCamera.

## 10.0 class GRSimpleFixedScreenCamera

### 10.1 constructor / Location() / LookAt() / Twist()

#### Purpose

A simplified camera is created. It behaves like a person looking into a television set that contains 3-D images. As you move around the TV, the objects should appear to remain stationary, although you can see parts of the top, sides and bottom of the objects. Of course, you can not see into the TV from the back of the set.

**Note:** While this class is useful to see the images displayed, it is most effective (and salient) when used with a stereoscopic display and a head position tracking device to update the camera location. (See GRSimpleStereoFixedScreenCamera).

#### Libraries

GROOP library (libgroop.a)

#### Syntax

```
#include <GRSimpleFixedScreenCamera.h>
GRSimpleFixedScreenCamera  gcamera;
void  gcamera.Location(location);
void  gcamera.LookAt(direction)
void  gcamera.Twist(rotation)

GRVertex  location;
GRVertex  direction;
float     rotation;
```

#### Description

GRSimpleFixedScreenCamera is a simplified GRCamera. Objects will appear to remain stationary (unless they are animated) as the camera is moved up, down, left and right. You can set the camera's location. The direction and rotation are maintained for compatibility with the GRSimpleCamera and GRSimpleStereoCamera classes.

#### Parameters

<b>location</b>	change the camera (observer's) location. The camera still faces the previous direction point. The default is to be at (0, 0, 8).
<b>direction</b>	the camera is turned to face towards this new point. The location of the camera does not change. The default direction is (0, 0, 0).
<b>rotation</b>	counter clockwise rotation of the camera. The rotation is along the line specified by the camera location and direction points.

#### Implementation Specifics

Default location and screen location values are taken from GRCamera.  
direction and rotation changes are ignored.

#### Related Information

See classes GLwindow, GLDisplay, and GRCamera, GRStereoCamera, GRSimpleCamera, GRSimpleStereoCamera and GRSimpleStereoFixedScreenCamera.

## 11.0 class GRSimpleStereoFixedScreenCamera

### 11.1 constructor / Location() / LookAt() / Twist() / EyeSeparation() / ParallaxConvergenceDepth()

#### Purpose

A simplified stereo camera is created. It behaves like a person looking into a television set that contains 3-D images. As you move around the TV, the objects should appear to remain stationary, although you can see parts of the top, sides and bottom of the objects. Of course, you can not see into the TV from the back of the set.

**Note:** While this class is most useful when using a large stereoscopic projection display, although it still works well with a stereoscopic monitor. To be most effective, head position tracking device should be used to continuously update the camera location.

#### Libraries

GROOP library (libgroop.a)

#### Syntax

```
#include <GRSimpleStereoFixedScreenCamera.h>
GRSimpleStereoFixedScreenCamera gcamera;
void gcamera.Location(location);
void gcamera.LookAt(direction)
void gcamera.Twist(rotation)
void gcamera.EyeSeparation(width)
void gcamera.ParallaxConvergenceDepth(depth)

GRVertex location;
GRVertex direction;
float rotation;
float width;
float depth;
```

#### Description

GRSimpleStereoFixedScreenCamera is a simplified GRStereoCamera. Objects will appear to remain stationary (unless they are animated) as the camera is moved up, down, left and right. You can set the camera's location. The direction and rotation functions are maintained for compatibility with the GRSimpleCamera and GRSimpleStereoCamera classes. ParallaxConvergenceDistance() has no effect and is maintained for compatibility with GRSimpleStereoCamera. EyeSeparation() has the same effect as in GRSimpleStereoCamera.

#### Parameters

<b>location</b>	change the camera (observer's) location. The camera still faces the previous direction point. The default is to be at (0, 0, 8).
<b>direction</b>	has no effect. The default direction is (0, 0, 0).
<b>rotation</b>	has no effect.
<b>width</b>	the distance between the two cameras.
<b>depth</b>	has no effect.

#### Implementation Specifics

| The simple stereo fixed screen camera consists of two GRCameras, each separated by the  
| EyeSeparation width. All computations for location are computed from the midpoint between the  
| two cameras.  
| Default location and direction values are taken from GRCamera.

| **Related Information**

| See classes GLwindow, GLDisplay, and GRCamera, GRStereoCamera, GRSimpleCamera,  
| GRSimpleStereoCamera and GRSimpleFixedScreenCamera.

## 11.2 ScreenPosition() / ScreenNormal() / ScreenUp() / ScreenWidth() / ScreenHeight() / NearClippingDistance() / FarClippingDistance()

### Purpose

Set various screen parameters for the stereo camera. The functions and parameters are the same as those for GRCamera.

### Libraries

GROOP library (libgroop.a)

### Syntax

```
#include <GRSimpleStereoFixedScreenCamera.h>
void    gcamera.ScreenPosition(position);
void    gcamera.ScreenNormal(normal);
void    gcamera.ScreenUp(up);
void    gcamera.ScreenWidth(width);
void    gcamera.ScreenHeight(height);
void    gcamera.NearClippingDistance(near);
void    gcamera.FarClippingDistance(near);

GRSimpleStereoFixedScreenCamera    gcamera;
GRVertex    position;
GRVertex    normal;
GRVertex    up;
float    width;
float    height;
float    near;
float    far;
```

### Description

The stereo camera is composed of two cameras. These functions permit changing both camera's parameters through a single function call.

### Parameters

See GRCamera and its member functions for details.

### Implementation Specifics

A call to any of these member functions will result in a similar call to each of the two cameras that make up the stereo camera.

### Related Information

See class GRCamera.

## 12.0 class GRLight

### 12.1 constructor

#### Purpose

Class constructor. Create a light for use in rendering.

#### Libraries

GROOP library (libgroop.a)

#### Syntax

```
#include <GRLight.h>  
GRLight light
```

#### Description

Create a default light source.

#### Implementation Specifics

The Reset() member function is called to set the initial values for the light.

#### Related Information

See classes GLwindow and GRDisplay.

## 12.2 Reset()

### Purpose

Reset a light's parameters.

### Libraries

GROOP library (libgroop.a)

### Syntax

```
#include <GRLight.h>
void light.Reset()

    GRLight light;
```

### Description

| Reset() removes any diffuse or ambient color that may be defined, and discards any spotlight lo-  
| cation information. The spotlight angle is set to 180.0 degrees, with an exponent of 0.0. The light  
| is set to be at infinite distance, and is not a local viewer (for shading calculations).

### Related Information

| This member function is called by the constructor. See Location(), Color(), Ambient(),  
| SpotDirection(), SpotLightAngleExponent(), and LocalViewer().

## 12.3 Color() / Ambient()

### Purpose

Change color of a light.

### Libraries

GROOP library (libgroop.a)

### Syntax

```
#include <GRLight.h>
void light.Color(color)
void light.Ambient(ambient)
```

```
    GRLight    light;
    GRrgb      color;
    GRrgb      ambient;
```

### Description

Lights contain two color components - its primary color and an ambient component. Color() changes the primary color component, and Ambient() changes the ambient component.

### Parameters

Both Color() and Ambient() take arguments of the type GRrgb. GRrgb accepts three floating point values as arguments, in the range 0.0 to 1.0, and represent the red, green, and blue components of a color (e.g., GRrgb(r, g, b) where r, g, b are in the range [0.0, 1.0]).

### Implementation Specifics

Color changes do not have an effect until the next time a window or display that uses the light is told to render ( Display() ).

### Related Information

See GRrgb, GLwindow and GRDisplay classes.

## 12.4 Location() / SpotDirection()

### Purpose

Change the location of a light.

### Libraries

GROOP library (libgroop.a)

### Syntax

```
#include <GRLight.h>
void light.Location(location)
void light.SpotDirection(direction)
```

```
GRLight    light;
GRvertex  location;
GRvertex  direction;
```

### Description

Changes the location of a light or the direction of a spotlight.

### Implementation Specifics

None of the other spotlight, local/global light parameters are modified.

Changes do not have an effect until the next time a window or display that uses the light is told to render ( Display() ).

### Related Information

See GLwindow and GRDisplay classes.

## | 12.5 SpotLightAngleExponent()

### **Purpose**

Set spotlight parameters.

### **Libraries**

GROOP library (libgroop.a)

### **Syntax**

```
| #include <GRLight.h>
| void light.SpotLightAngleExponent (angle, exponent)
|
|     GRLight   light;
|     float     angle
|     float     exponent
```

### **Description**

Change the spotlight angle and light loss exponent. These parameters are used when the light is designated as a spotlight.

### **Implementation Specifics**

```
| None of the other light parameters are affected. These parameters have no effect unless the
| SpotDirection() has been defined.
|
| Changes do not have an effect until the next time a window or display that uses the light is told
| to render ( Display() ).
```

### **Related Information**

```
| See SpotDirection(), GLwindow and GRDisplay classes.
```

## | **12.6 LocalViewer()**

### | **Purpose**

| Modify the lighting calculations to make the surfaces appear more realistic, at the cost of slower rendering speed.

### | **Libraries**

| GROOP library (libgroop.a)

### | **Syntax**

```
| #include <GRLight.h>
| void light.LocalViewer(boolean)
|
|     GRLight    light;
|     int        boolean;
```

### | **Description**

| Lighting calculations on a vertex-by-vertex basis can be done more or less accurately depending on whether the viewer is considered local or not. It is a speed/accuracy trade-off. The boolean argument can be TRUE or FALSE. When FALSE, the lights are considered to be at infinite distance for the purposes of computing lighting effects. This is the faster mode of operation. When TRUE, a more realistic rendering can be performed because the location of the viewer is taken into account. However, rendering speed is adversely affected.

### | **Parameters**

| boolean is TRUE or FALSE.

### | **Related Information**

| See the Graphics Programming Concepts manual (SC23-2208) for further details.

## 13.0 class GRGeometricObject

This class is an "abstract class" upon which many other classes are derived. This class contains basic member functions used by geometric objects -- transformations and material properties.

### 13.1 Rotate / Scale() / Translate() / GenericTrans()

#### Purpose

3-D transformations.

#### Libraries

GROOP library (libgroop.a)

USL C++ Standard Components library (lib++.a)

#### Syntax

```
#include <GRGeometricObject.h>
void gobject.RotateX(rotation)
void gobject.RotateY(rotation)
void gobject.RotateZ(rotation)
void gobject.Scale(scaleX, scaleY, scaleZ)
void gobject.Translate(translateX, translateY, translateZ)
void gobject.GenericTrans(matrix)
```

```
GRGeometricObject  gobject;
float               rotation;
float               scaleX, scaleY, scaleZ;
float               translateX, translateY, translateZ;
GRMatrix           matrix;
```

#### Description

Rotate, Scale, Translate or perform an arbitrary transformation on an object. When an object is rendered, the transformations are applied in the order specified.

#### Parameters

Rotations are specified in degrees.

Translations are specified in relative coordinates. That is, two successive translations will result in the composition of the translations, not just the application of only one of the translations.

#### Implementation Specifics

Generally, after the geometric object is drawn by the display (e.g. GLwindow) the transformations are discarded. The exceptions are when a SaveTransforms() is performed (see below) or ResetTransformsOnDisplay() is set to FALSE.

Transformations are maintained as a list. When a SaveTransforms() is performed, the transforms on the list up to the SaveTransforms() are computed and saved as a single generic transformation (4x4 matrix). Subsequent transformations are kept on the list. This approach simplifies modelling, animation and improves overall performance of GROOP.

#### Related Information

See SaveTransforms(), ResetTransforms(), RemoveTransforms(), SetScale(), SetRotate(), SetTranslate() and SetScaleRotateTranslate() member functions.

See ResetTransformsOnDisplay() and GetResetTransformsOnDisplay() member functions.

Also, ResetTransformsOnDisplay in GLwindow.

Also see classes derived from GRGeometricObject.

## 13.2 SetRotate() / SetScale() / SetTranslate() / SetScaleRotateTranslate()

### Purpose

Static 3-D transformations.

### Libraries

GROOP library (libgroop.a)

### Syntax

```
#include <GRGeometricObject.h>
void gobject.SetScale(scaleX, scaleY, scaleZ)
void gobject.SetRotate(rotateX, rotateY, rotateZ)
void gobject.SetTranslate(translateX, translateY, translateZ)
void gobject.SetScaleRotateTranslate(scaleX, scaleY, scaleZ,
                                     rotateX, rotateY, rotateZ,
                                     translateX, translateY, translateZ)

GRGeometricObject  gobject;
float               rotateX, rotateY, rotateZ;
float               scaleX, scaleY, scaleZ;
float               translateX, translateY, translateZ;
```

### Description

Set a static transformation (rotate, scale or translate) that is *not* reset after each display of a geometric object. These functions provide an easy to use method of building models. Transformations are applied in the following order (if present): scaling, rotating (rotateX then rotateY then rotateZ), and then translation.

### Parameters

Rotations are specified in degrees.

Scaling values must not be zero.

Translations are specified relative to the origin.

### Implementation Specifics

If any of the scaling values are zero (0.0), then scaling is ignored. If all of the rotations are zero (0.0), then rotations are ignored. Rotations are applied in the order of x then y then z. If all translations are zero (0.0), then translations are ignored.

### Related Information

See Scale(), Translate(), RotateX(), RotateY(), and RotateZ() member functions.

Also see classes derived from GRGeometricObject.

## 13.3 SaveTransforms() / RemoveTransforms() / ResetTransforms()

### Purpose

Perform operations on the list of transformations for the geometric object.

### Libraries

GROOP library (libgroop.a)  
USL C++ Standard Components library (lib++.a)

### Syntax

```
#include <GRGeometricObject.h>
void gobject.SaveTransforms()
void gobject.RemoveTransforms()
void gobject.ResetTransforms()

GRGeometricObject gobject;
```

### Description

Transformations (RotateX(), RotateY(), RotateZ(), Scale(), Translate(), GenericTrans()) are saved as a list of operations to be performed. To simplify modelling and animation, and improve GROOP performance, the list of transformations can be computed and reduced to a single 4x4 transformation matrix and saved for future use. To save the list of transformations, SaveTransforms() is called. The computation of the saved transformation matrix is as follows:

1. Use the previously saved transformation matrix, if it exists. If not, use an identity matrix.
2. Apply all transformations on the list (RotateX, RotateY, RotateZ, Scale, Translate, GenericTrans) to this matrix until a SaveTransforms is reached.
3. When a SaveTransforms is found on the transformation list, save the current transformation matrix as the new saved transformation matrix.

Multiple SaveTransforms() calls may be made on an object.

SaveTransforms() is typically used to save a set of transforms needed to construct and preserve a model in the correct orientation, scale and translation. If the model is to be modified (e.g., for animation) subsequent transformations can be added after the saved transformations.

Two member functions are defined to reset transformations. The first, *ResetTransforms()*, removes all transforms that have not been saved by SaveTransforms(). This member function is usually called by a window or display after an object has been rendered. *The net effect is that any transformations applied to an object for the purposes of animation are removed.*

The second technique for resetting transformations is *RemoveTransforms()*, which calls ResetTransforms() *and* removes any previously saved transformation matrix computed by SaveTransforms().

### Implementation Specifics

Transformations and SaveTransforms are maintained in a single list. SaveTransforms are not computed until a Display() member function is called for a window or display that is trying to render the geometric object. SaveTransforms() slows down rendering during the first Display() call to a window or display. Subsequent Display() calls will be faster.

### Related Information

See transformation member functions for GRGeometricObject (RotateX, RotateY, RotateZ, Scale, Translate, GenericTrans). Also see classes derived from GRGeometricObject, and GLwindow and GRDisplay.

## 13.4 ResetTransformsOnDisplay() / GetResetTransformsOnDisplay()

### Purpose

Inform a window / display whether the transforms in the current geometric object should be deleted after the object is rendered.

### Libraries

GROOP library (libgroop.a)  
USL C++ Standard Components library (lib++.a)

### Syntax

```
#include <GRGeometricObject.h>
int  gobject.ResetTransformsOnDisplay(boolean)
int  gobject.GetResetTransformsOnDisplay(boolean)

GRGeometricObject  gobject;
int                boolean;    // TRUE/FALSE
```

### Description

When a window / display object renders a geometric object, it usually calls the geometric object's ResetTransforms() member function. ResetTransformsOnDisplay(TRUE) is the default, so when a window / display renders the geometric object, ResetTransforms() is called. However, if an object calls ResetTransformsOnDisplay(FALSE), then the display / window will not call ResetTransforms() for that object.

Both ResetTransformsOnDisplay() and GetResetTransformsOnDisplay() return the current setting (TRUE or FALSE).

### Related Information

See ResetTransformsOnDisplay() member function for class GLwindow.

## | **13.5 ReplaceTransforms()**

### | **Purpose**

| Replace the current list of transforms with a new list. The old list is deleted.

### | **Libraries**

| GROOP library (libgroop.a)

| USL C++ Standard Components library (lib++.a)

### | **Syntax**

```
| #include <GRGeometricObject.h>
| int  gobject.ReplaceTransforms(transformsList)
|
|     GRGeometricObject  gobject;
|     List_of_p(GRGeoTransforms) transformsList;
```

### | **Description**

| A geometric object's list of transforms can be replaced with a single function call rather than  
| having to manually reset the list of transforms and add the new transforms to the list one at a time.

### | **Parameters**

| transformsList is a list of pointers to GRGeoTransforms objects.

### | **Related Information**

| See USL SC List class and GRGeoTransforms class.

## 13.6 material

### Purpose

A member variable of `GRGeometricObject` that contains the material properties, such as color, of the geometric object.

### Libraries

GROOP library (`libgroop.a`)

### Syntax

```
#include <GRGeometricObject.h>
GRMaterialProperties  gobject.material
```

### Description

Each geometric object contains a set of material properties. Through `GRMaterialProperties` member functions, the various material properties can be set or reset.

### Related Information

See `GRMaterialProperties` and `GRComposite` classes.

## 14.0 class GRMaterialProperties

This class describes material properties of surfaces and is used during the rendering process. These properties include such attributes as colors and shininess.

This class is primarily used by GRGeometricObject to set the *material* member variable.

### 14.1 Emission() / Ambient() / Diffuse() / Specular() / Shininess() / Reset() / TextureMap()

#### Purpose

Set (reset) the material properties of renderable objects.

#### Libraries

GROOP library (libgroop.a)

#### Syntax

```
#include <GRMaterialProperties.h>
void material.Emission(rgb)
void material.Ambient(rgb)
void material.Diffuse(rgb)
void material.Specular(rgb)
void material.Shininess(shininess)
+ void material.TextureMap(map)
void material.Reset()

GRMaterialProperties material;
GRrgb rgb;
+ GRTextureMap* map;
float shininess;
```

#### Description

- Different color attributes can be assigned: Emission, Ambient, Diffuse, and Specular.
- The shininess of the material can be set.
- + • A texture map (GRTextureMap object) can be assigned to an object.
- All material properties can be reset to default values by issuing a call to Reset(). See Error Codes below for resetting individual attributes.

#### Parameters

Each of the color attributes accepts an GRrgb value ( GRrgb(r, g, b) ), where r, g and b are in the range [0.0, 1.0].

Shininess of the material can be set with values in the range [0.0, 1.0].

- + Texture maps are defined by the class GRTextureMap. They are rectangular arrays of rgb images or color blending information that can be applied to object surfaces.

#### Error Codes

If a value specified in GRrgb or shininess is outside the range [0.0, 1.0], then the material attribute is reset.

#### Implementation Specifics

The default material properties are defined by each window or display class.

- + Texture maps are initially supported only on GRPolygon objects. All other objects ignore the texture mapping information.
- + Texture mapping is initially supported on a subset of the Silicon Graphics computers. If texture mapping is not supported, then the mapping information is ignored.

**Related Information**

- + See GRrgb, GRTextureMap, GRGeometricObject and GLwindow.

## 15.0 classes GRPoint and GRPointList

### 15.1 GRPoint

#### Purpose

Define a 3-D point with an optional normal.

#### Libraries

GROOP library (libgroop.a)

#### Syntax

```
#include <GRPoint.h>
GRPoint point;
GRPoint point(vertex);
GRPoint point(x, y, z);
GRPoint point(vertex, normal);
GRPoint point(x, y, z, normalX, normalY, normalZ);

GRVertex vertex, normal;
float x, y, z, normalX, normalY, normalZ;
```

#### Description

3-D points can be created, with or without normals. The default point is at (0.0, 0.0, 0.0) with a normal of (1.0, 0.0, 0.0). When a normal is not supplied, the default normal used is (1.0, 0.0, 0.0).

GRPoint is derived from GRGeometricObject, so instances are renderable. They can be used as arguments to Add() member functions for windows, displays and objects of type GRComposite.

#### Parameters

Arguments can be specified as either GRVertex objects, or through the use of three floating point values for each vertex and normal.

#### Implementation Specifics

When not specified in the constructor, default normals are defined to be (1.0, 0.0, 0.0).

GRPoint objects should not be added to more than one GRComposite or GRPointList, otherwise results may be unpredictable (e.g., core dump).

#### Related Information

See GRGeometricObject, GRPointList, GRPointsReader, GRSimplePointsReader, GRComposite, GLwindow, and GRDisplay.

## 15.2 GRPointList

### Purpose

Describe a list of points.

### Libraries

GROOP library (libgroop.a)

USL C++ Standard Components library (lib++.a)

### Syntax

```
#include <GRPointList.h>
GRPointList list;
GRPointList list(reader);
GRPointList list(pointRef);
void list.Add(pointRef);
void list.Add(point);

GRPointList list;
GRPointsReader* reader;
GRPoint point;
GRPoint* pointRef;
```

### Description

A list of points is created by the constructor. If a pointer to a GRPoint is passed to the constructor, then the GRPoint is added to the list. If a pointer to a GRPointsReader is passed, then the constructor calls the reader defined in the GRPointsReader object. The GRPointsReader is an object whose purpose is to read multiple points (e.g., from a file) and add them to the GRPointsList object.

Add() takes a point or a reference to a point as an argument. The point is added to the points list. GRPointList objects can be added to a GRComposite object through the Add() member function.

### Parameters

<b>point</b>	a GRPoint object.
<b>pointRef</b>	the address of a GRPoint object. pointRef should not be added to any other objects (e.g., GRPointLists or GRComposites).
<b>reader</b>	a GRPointsReader object. The GRPointList constructor calls a function in the GRPointsReader object (ReadFile()) to create (read) a set of points and add them to the GRPointList object.

### Implementation Specifics

- When GRPointList calls its destructor, each GRPoint in the list will be deleted as well.
- If a large number of points need to be rendered, then GRPointList object is faster than if the GRPoint objects were part of a GRComposite object.
- Only the material properties and transformations for the GRPointList object are used. The material properties and transformations for each point in the GRPointList are ignored during rendering.
- GRPoint objects should not be added to more than one GRComposite or GRPointList, otherwise results may be unpredictable (e.g., core dump).
- The GRPointsReader object is *not* deleted by the constructor or destructor.

### **Related Information**

| See `GRGeometricObject`, `GRPoint`, `GRComposite`, `GRPointsReader`, `GRSimplePointsReader`, `GLwindow`, and `GRDisplay`.

## 16.0 classes **GRLine** and **GRLineList**

### 16.1 **GRLine**

#### **Purpose**

Define a polyline.

#### **Libraries**

GROOP library (libgroop.a)

#### **Syntax**

```
#include <GRLine.h>
GRLine   line;
GRLine   line(vertex);
GRLine   line(vertex, normal);
GRLine   line(width);
void line.Add(vertex);
void line.Add(vertex, normal);
void line.Width(width);
int  line.GetWidth();

GRLine   line;
GRVertex vertex, normal;
int      width;
```

#### **Description**

**GRLine** is a list of vertices (normals are optional) used to define a polyline. The constructor accepts an optional vertex (and normal) as the first point in the polyline. Additional vertices for the polyline are added through the `Add()` member function (inherited from class `GRVertexList`).

The width of the line can optionally be specified in the constructor or through the `Width()` member function. Width is specified in pixels.

`GetWidth()` member function returns the width of the line in pixels.

#### **Parameters**

When vertices are being specified in the constructor or member functions, the normals are optional.

The width of the line must be greater or equal to 1, and represents the width in pixels.

#### **Implementation Specifics**

- **GRLine** inherits from `GRVertexList`. `Add()` member functions can be found in the `GRVertexList` class.
- **GRLine** objects should not be added to more than one `GRLineList` or `GRComposite`, otherwise results may be unpredictable (e.g., core dump).
- The default line width is 1.
- Zero or negative line width specifications will be ignored.

#### **Related Information**

See `GRGeometricObject`, `GRVertexList`, `GRLineList`, `GRLinesReader`, `GRSimpleLinesReader`, `GRComposite`, `GLwindow`, and `GRDisplay`.

## 16.2 GRLineList

### Purpose

Describe a list of polylines.

### Libraries

GROOP library (libgroop.a)  
USL C++ Standard Components library (lib++.a)

### Syntax

```
#include <GRLineList.h>
GRLineList list;
GRLineList list(line);
GRLineList list(reader);
void list.Add(line);

GRLineList list;
GRLine* line;
GRLinesReader* reader;
```

### Description

A list of polylines is created by the constructor. If a pointer to a GRLine is passed to the constructor, then the GRLine is added to the list.

If a GRLinesReader is passed to the constructor, the ReadFile() function of the GRLinesReader is called and it adds lines to the GRLineList (typically by reading the descriptions of the polylines from a file).

Add() takes the polyline referenced by the argument and adds it to the list.

GRLineList objects can be added to GRComposite objects.

### Parameters

Arguments are GRLine objects, and are passed by reference.

### Implementation Specifics

- When GRLineList calls its destructor, each GRLine in the list will be deleted as well.
- If a large number of polyline need to be rendered, then GRLineList object will display the GRLine objects faster than if added to a GRComposite object.
- Only the material properties and transformations for the GRLineList object are used. The material properties and transformations for each line in the GRLineList are ignored during rendering.
- GRLine objects should not be added to more than one GRLineList or GRComposite, otherwise results may be unpredictable (e.g., core dump).
- GRLineList objects should not be added to more than one GRComposite, otherwise results may be unpredictable (e.g., core dump).
- The GRLinesReader object is *not* deleted by the constructor or destructor.

### Related Information

See GRGeometricObject, GRLine, GRComposite, GRLinesReader, GRSimpleLinesReader, GLwindow, and GRDisplay.

## 17.0 classes GRTriMesh and GRTriMeshList

### 17.1 GRTriMesh

#### Purpose

Define a triangle mesh (TriMesh), also know as triangle strips.

#### Libraries

GROOP library (libgroop.a)

#### Syntax

```
#include <GRTriMesh.h>
GRTriMesh    trimesh;
GRTriMesh    trimesh(vertex, normal);
void trimesh.Add(vertex, normal);

GRTriMesh    trimesh;
GRVertex     vertex, normal;
```

#### Description

GRTriMesh is a list of verticies used to define the points in a mesh (or strip) of triangles. The constructor accepts an optional (vertex, normal) pair as the first point in the trimesh. Additional points in the polyline are added to the polyline through the Add() member function (inherited from the class GRVertexList).

GRTriMesh objects can be added to GRTriMeshList, GRComposite and window or display objects.

#### Implementation Specifics

- GRTriMesh inherits from GRVertexList. Add() member function can be found in the GRVertexList class.
- GRTriMesh objects should not be added to more than one GRTriMeshList or GRComposite, otherwise results may be unpredictable (e.g., core dump).

#### Related Information

| See GRGeometricObject, GRVertexList, GRTriMeshList, GRTriMeshReader,  
| GRSimpleTriMeshReader, GRComposite, GLwindow, and GRDisplay.

## 17.2 GRTriMeshList

### Purpose

Describe a list of Triangle Meshes (GRTriMesh).

### Libraries

GROOP library (libgroop.a)

USL C++ Standard Components library (lib++.a)

### Syntax

```
#include <GRTriMeshList.h>
GRTriMeshList list;
GRTriMeshList list(trimesh);
GRTriMeshList list(reader);
void list.Add(trimesh);

GRTriMeshList list;
GRTriMesh* trimesh;
GRTriMeshReader* reader;
GRTriMeshReader* trimesh;
```

### Description

A list of triangle meshes is created by the constructor. If a pointer to a GRTriMesh is passed to the constructor, then the GRTriMesh is added to the list.

If a GRTriMeshReader is passed to the constructor, the ReadFile() function of the GRTriMeshReader is called and it adds GRTriMesh's to the GRTriMeshList (typically by reading the descriptions of the GRTriMesh's from a file).

Add() takes the triangle mesh referenced by the argument and adds it to the list.

GRTriMeshList objects can be added to GRComposite objects, and windows or displays.

### Parameters

<b>trimesh</b>	GRTriMesh objects, passed by reference.
<b>reader</b>	a GRTriMeshReader object. The GRTriMeshList constructor calls a function in the GRTriMeshReader object (ReadFile()) to create (read) a set of GRTriMesh objects and add them to the GRTriMesh object.

### Implementation Specifics

- When GRTriMeshList calls its destructor, each GRTriMesh in the list will be deleted as well.
- If a large number of triangle meshes need to be rendered, then GRTriMeshList object will display them faster than if the GRTriMesh objects were added to a GRComposite object.
- Only the material properties and transformations for the GRTriMeshList object are used. The material properties and transformations for each triangle mesh in the GRTriMeshList are ignored during rendering.
- GRTriMesh objects should not be added to more than one GRTriMeshList or GRComposite, otherwise results may be unpredictable (e.g., core dump).
- GRTriMeshList objects should not be added to more than one GRComposite, otherwise results may be unpredictable (e.g., core dump).
- The GRTriMeshReader object is *not* deleted by the constructor or destructor.

### Related Information

| See `GRGeometricObject`, `GRTriMesh`, `GRTriMeshReader`, `GRSimpleTriMeshReader`,  
`GRComposite`, `GLwindow`, and `GRDisplay`.

## 18.0 classes GRPolygon and GRPolygonList

### 18.1 GRPolygon

#### Purpose

Define a polygon.

#### Libraries

GROOP library (libgroop.a)

#### Syntax

```
#include <GRPolygon.h>
GRPolygon polygon;
GRPolygon polygon(vertex, normal);
void polygon.Add(vertex, normal);

GRPolygon polygon;
GRVertex vertex, normal;
```

#### Description

GRPolygon is a list of vertices used to define a polygon. The constructor accepts an optional (vertex, normal) pair as first point in the polygon. Additional points in the polygon are added to the polygon through the Add() member function (inherited from the GRVertexList class).

GRPolygon objects can be added to GRComposite objects and window or display objects.

#### Implementation Specifics

- GRPolygon inherits from GRVertexList. Add() member function can be found in the GRVertexList class.
- GRPolygon objects should not be added to more than one GRPolygonList or GRComposite, otherwise results may be unpredictable (e.g., core dump).

#### Related Information

| See GRGeometricObject, GRVertexList, GRPolygonList, GRPolygonReader,  
| GRSimplePolygonReader, GRComposite, GLwindow, and GRDisplay.

## 18.2 GRPolygonList

### Purpose

Describe a list of Polygons (GRPolygon).

### Libraries

GROOP library (libgroop.a)

USL C++ Standard Components library (lib++.a)

### Syntax

```
#include <GRPolygonList.h>
GRPolygonList list;
GRPolygonList list(polygon);
GRPolygonList list(reader);
void list.Add(polygon);

GRPolygonList list;
GRPolygon* polygon;
```

### Description

A list of polygons is created by the constructor. If a pointer to a GRPolygon is passed to the constructor, then the GRPolygon is added to the list.

If a GRPolygonReader is passed to the constructor, the ReadFile() function of the GRPolygonReader is called and it adds GRPolygon's to the GRPolygonList (typically by reading the descriptions of the GRPolygon's from a file).

Add() takes the polygon referenced by the argument and adds it to the list.

GRPolygonList objects can be added to GRComposite's, and windows or displays.

### Parameters

<b>polygon</b>	GRPolygon objects, passed by reference.
<b>reader</b>	a GRPolygonReader object. The GRPolygonList constructor calls a function in the GRPolygonReader object (ReadFile()) to create (read) a set of GRPolygon objects and add them to the GRPolygon object.

### Implementation Specifics

- When GRPolygonList calls its destructor, each GRPolygon in the list will be deleted as well.
- If a large number of polygons need to be rendered, then using a GRPolygonList object will result in faster display than if the GRPolygon objects were put into a GRComposite object.
- Only the material properties and transformations for the GRPolygonList object are used. The material properties and transformations for each polygon in the GRPolygonList are ignored during rendering.
- GRPolygon objects should not be added to more than one GRPolygonList or GRComposite, otherwise results may be unpredictable (e.g., core dump).
- GRPolygonList objects should not be added to more than one GRComposite, otherwise results may be unpredictable (e.g., core dump).
- The GRPolygonReader object is *not* deleted by the constructor or destructor.

### Related Information

| See GRGeometricObject, GRPolygon, GRComposite, GRPolygonReader,  
| GRSimplePolygonReader GLwindow, and GRDisplay.

## 19.0 class GRText

### 19.1 GRText

#### Purpose

Define a 2-D text string.

#### Libraries

GROOP library (libgroop.a)

#### Syntax

```
#include <GRText.h>
GRText text(message, font);
void text.SetText(message);
char* text.GetText();
void text.SetFont(font);
char* text.GetFont();

GRText text;
char* message, font;
```

#### Description

Two dimensional text (usually using raster or outline fonts) can be rendered into a 3-D scene. GRText objects are GRGeometric objects, so they can have material attributes defined, and geometric transforms can be performed on the origin of where the text string should be rendered. However, most implementations of 2-D text will not properly scale or rotate the text as would usually be expected of geometric objects.

#### Parameters

The message and font variables point to null terminated character strings.

The font is renderer dependent. See implementation specifics below.

#### Implementation Specifics

In GLwindow on AIX/6000, raster fonts are obtained from the X Windows server. So, the font specified must be a valid X Windows font.

#### Related Information

See GRGeometricObject, GRComposite, GLwindow, and GRDisplay.

## 20.0 class GRComposite

### 20.1 GRComposite

#### Purpose

Create a hierarchical composition (aggregation) of renderable objects consisting of objects derived from GRGeometricObject or other GRComposite objects.

#### Libraries

GROOP library (libgrop.a)  
USL C++ Standard Components library (lib++.a)

#### Syntax

```
#include <GRComposite.h>
GRComposite composite;
GRComposite composite(reader);
void composite.Add(gobject);
void composite.Delete(gobject);
void composite.DeleteOnReset(bool);
int composite.DeleteOnReset();
void composite.ResetObjects();

GRComposite      composite.
GRCompositeReader reader;
GRGeometricObject gobject;
int              boolean; // TRUE or FALSE
```

#### Description

GRComposite objects are heterogeneous lists of renderable objects, either derived from GRGeometricObject or GRComposite. GRGeometricObject subclasses include GRTriMesh, GRPolygon, GRLine, among others. GRComposite objects are containers for modelling larger or aggregated objects. For example, a group of objects can be added to the composite, and then transformed (e.g., RotateX(), RotateY(), RotateZ(), Scale(), Translate()) and given material properties. The transformations are applied to all of the objects contained in the composite, and material properties are inherited by these objects.

If a GRCompositeReader is passed to the constructor, the ReadFile() function of the GRCompositeReader is called and it adds GRGeometricObjects to the GRComposite (typically by reading the descriptions of the objects from a file).

While an object may inherit material properties from a GRComposite, it can override those properties by defining its own material properties.

To render a GRComposite, pass it to a window or display's Add() member function.

Objects are added to the composite using the Add() member function, and removed by using the Delete() member function.

By default, ResetObjects() member function removes all objects from the composite. To prevent the objects from being deleted, call DeleteOnReset() with a value of FALSE. Calling DeleteOnReset() with a value of TRUE will cause the ResetObjects() member function to remove all objects from the composite. DeleteOnReset() (without any arguments) returns the current setting (TRUE or FALSE).

**Note:** The destructor for GRComposite() calls ResetObjects().

#### Parameters

reader is a `GRCompositeReader` object. The `GRComposite` constructor calls a function in the `GRCompositeReader` object (`ReadFile()`) to create (read) a set of objects and add them to the `GRComposite` object.

`gobject` is usually an instance of an object of a class derived from the class `GRGeometricObject`, not an instance of `GRGeometricObject`.

`bool` is either `TRUE` or `FALSE`.

### Implementation Specifics

- When `GRComposite` calls its destructor, all of the objects added (via the `Add()` member function) may be deleted as well. See the description of `DeleteOnReset()` and `ResetObjects()` above.
- If `DeleteOnReset()` is `TRUE`, then `GRComposite` objects should not be added to more than one other `GRComposite`, otherwise results may be unpredictable (e.g., core dump).
- If a large number of points, lines, trimeshes or polygons need to be rendered, then consider using `GRPointList`, `GRLineList`, `GRTriMeshList` or `GRPolygonList` because they are faster when rendering a homogenous set of objects.
- The `GRCompositeReader` is not deleted by the constructor or destructor.

**Note:** Inheritance of material properties does not always work currently in the initial implementation. This will be corrected in a future release of GROOP.

### Related Information

See `GRGeometricObject`, `GRPointList`, `GRLineList`, `GRTriMeshList`, `GRPolygonList`, `GRCompositeReader`, `GLwindow`, and `GRDisplay`.

## | **21.0 file readers (GRPointsReader, GRLinesReader, | GRTriMeshReader, GRPolygonReader, GRCompositeReader)**

| GROOP is not tied to any one file format. Files are read in through subclasses of GRPointsReader, GRLinesReader, GRTriMeshReader, GRPolygonReader, and GRCompositeReader. (See GRFileReaders.h for the class definitions.) These classes are virtual base classes. The classes are typically used to read in data from files and create composite objects (homogeneous and heterogenous).

| Simple flat file readers are implemented in classes GRSimplePointsReader, GRSimpleLinesReader, GRSimpleTriMeshReader, and GRSimplePolygonReader.

## 21.1 GRSimplePointsReader

### Purpose

Read in a file of points and add them to a GRPointsList object.

### Libraries

GROOP library (libgroop.a)

### Syntax

```
#include <GRSimpleFileReaders.h>
GRSimplePointsReader  reader(filename);
void reader.ReadFile(pointslist);

GRSimplePointsReader  reader;
GRPointsList          pointslist;
char*                 filename;
```

### Description

The constructor saves the file name passed as a char\*. When ReadFile() is called, the file is opened and points data is read. Each point read from the file is added to the GRPointsList object passed as an argument to ReadFile(). Once the file is read, it is closed.

The ascii file format is as follows:

<b>file label</b>	A single token (blank/tab/<cr> delimited) that describes the file.
<b>count</b>	Number of points in the file.
<b>point(s)</b>	Each point is three floating point values (blank/tab/<cr> delimited).

### Parameters

<b>filename</b>	a null terminated string that contains the name of the file to be read. This string is copied by the constructor.
<b>pointslist</b>	a GRPointsList*, pointing to a GRPointsList. Points created by ReadFile() are added to pointslist (e.g., pointslist->Add(newpoint)).

### Implementation Specifics

If the files does not exist, or can not be opened for reading, then no points are added to the pointslist.

If an incorrect number of points is specified as the second argument in the file, then results may be unpredictable.

After the file is read, it is closed.

### Related Information

See GRPointsList, GRSimpleLinesReader, GRSimpleTriMeshReader, GRSimplePolygonReader, GRFileReaders.h, GRSimpleFileReaders.h, GRSimpleFileReaders.C, GRLineList, GRTriMeshList, GRPolygonList and GRPointsList.

## 21.2 GRSimpleLinesReader

### Purpose

Read in a file of lines and add them to a GRLineList object.

### Libraries

GROOP library (libgroop.a)

### Syntax

```
#include <GRSimpleFileReaders.h>
GRSimpleLinesReader  reader(filename);
void reader.ReadFile(lineslist);

GRSimpleLinesReader  reader;
GRLineList           linelist;
char*                filename;
```

### Description

The constructor saves the file name passed as a char\*. When ReadFile() is called, the file is opened and lines (polylines) data is read. Each line read from the file is added to the GRLineList object passed as an argument to ReadFile(). Once the file is read, it is closed.

The ascii file format is as follows:

<b>file label</b>	A single token (blank/tab/<cr> delimited) that describes the file.
<b>count</b>	Number of lines in the file.
<b>line(s)</b>	For each line in the file, there is a count of the number of points in the polyline, followed by the points. Each point is three floating point values (blank/tab/<cr> delimited).

### Parameters

<b>filename</b>	a null terminated string that contains the name of the file to be read. This string is copied by the constructor.
<b>pointslist</b>	a GRLineList*, pointing to a GRLineList. Lines created by ReadFile() are added to linelist (e.g., linelist->Add(newline)).

### Implementation Specifics

If the files does not exist, or can not be opened for reading, then no lines are added to the linelist.  
If an incorrect number of lines or points is specified in the file, then results may be unpredictable.  
After the file is read, it is closed.

### Related Information

See GRLineList, GRSimplePointsReader, GRSimpleTriMeshReader, GRSimplePolygonReader, GRFileReaders.h, GRSimpleFileReaders.h, GRSimpleFileReaders.C, GRLineList, GRTriMeshList, GRPolygonList and GRPointsList.

## 21.3 GRSimpleTriMeshReader

### Purpose

Read in a file of triangle meshes and add them to a GRTriMeshList object.

### Libraries

GROOP library (libgroop.a)

### Syntax

```
#include <GRSimpleFileReaders.h>
GRSimpleTriMeshReader reader(filename);
void reader.ReadFile(trimeshlist);

GRSimpleTriMeshReader reader;
GRTriMeshList trimeshlist;
char* filename;
```

### Description

The constructor saves the file name passed as a char\*. When ReadFile() is called, the file is opened and triangle mesh data is read. Each triangle mesh read from the file is added to the GRTriMeshList object passed as an argument to ReadFile(). Once the file is read, it is closed.

The ascii file format is as follows:

<b>file label</b>	A single token (blank/tab/<cr> delimited) that describes the file.
<b>action</b>	a single int that describes the purpose of the six floating point values that follow. The following constants are defined in GRSimpleFileReaders.h.
<b>addPoint</b>	The next six floating point values are to be interpreted as a vertex/normal pair, and are added to the current triangle mesh.
<b>swapPoint</b>	Tells the renderer to perform the equivalent of GL's swaptmesh().
<b>lastPoint</b>	The next six floating point values are to be interpreted as a vertex/normal pair, and are added to the current triangle mesh. This is the last point in the current triangle mesh. Subsequent points are added to a new triangle mesh.
<b>colorList</b>	The next three floating point values are used as the color of the GRTriMeshList. The last three floating point values are ignored.

### Parameters

<b>filename</b>	a null terminated string that contains the name of the file to be read. This string is copied by the constructor.
<b>trimeshlist</b>	a GRTriMeshList*, pointing to a GRTriMeshList. Triangle meshes created by ReadFile() are added to trimeshlist (e.g., trimeshlist->Add(newtrimesh)).

### Implementation Specifics

If the files does not exist, or can not be opened for reading, then no triangle meshes are added to the trimeshlist.

| If the file does not adhere to the format specified above, then results may be unpredictable.  
| After the file is read, it is closed.

| **Related Information**

| See `GRTriMeshList`, `GRSimplePointsReader`, `GRSimpleLinesReader`, `GRSimplePolygonReader`,  
| `GRFileReaders.h`, `GRSimpleFileReaders.h`, `GRSimpleFileReaders.C`, `GRLineList`, `GRTriMeshList`,  
| `GRPolygonList` and `GRPointsList`.

## 21.4 GRSimplePolygonReader

### Purpose

Read in a file of indexed polygons and add them to a GRPolygonList object.

### Libraries

GROOP library (libgroop.a)

### Syntax

```
#include <GRSimpleFileReaders.h>
GRSimplePolygonReader reader(filename, normalType);
void reader.ReadFile(polygonlist);

GRSimplePolygonReader reader;
GRPolygonList polygonlist;
char* filename;
int normalType;
```

### Description

The constructor saves the file name passed as a char\*. When ReadFile() is called, the file is opened and indexed polygon data is read. Polygons read from the file are added to the GRPolygonList object passed as an argument to ReadFile(). Once the file is read, it is closed.

The ascii file format is as follows:

<b>file label</b>	A single token (blank/tab/<cr> delimited) that describes the file.
<b>point count</b>	number of points to be read from the file
<b>polygon count</b>	number of polygons (sets of indexes) to be read from the file
<b>points</b>	each point is a set of three floating point values (blank/tab/<cr> delimited). The number of these triplets is specified by the "point count".
<b>index list</b>	each polygon is specified by an indexed list. The index list specifies the index of three or more points that compose a polygon. By using the index to look up the points, the polygons are constructed and face normals are computed.
	<b>index count</b> the number of index values for the current polygon. The index values follow the index count.
	<b>index values</b> a list of indices to look up the next point in the polygon.

### Parameters

<b>filename</b>	a null terminated string that contains the name of the file to be read. This string is copied by the constructor.
<b>normalType</b>	Either face normals or point normals can be used when constructing the polygons. If the value is TRUE, face normals are used. If FALSE, point normals are computed (an averaged normals for all faces to which the point belongs).
<b>polygonlist</b>	a GRPolygonList*, pointing to a GRPolygonList. Polygons created by ReadFile() are added to polygonlist (e.g., polygonlist->Add(newpolygon)).

| **Implementation Specifics**

|       If the file does not exist, or can not be opened for reading, then no polygons are added to the polygonlist.

|       If the file does not adhere to the format specified above, then results may be unpredictable.

|       After the file is read, it is closed.

| **Related Information**

|       See GRPolygonList, GRSimplePointsReader, GRSimpleLinesReader, GRSimpleTriMeshReader, GRFileReaders.h, GRSimpleFileReaders.h, GRSimpleFileReaders.C, GRLineList, GRTriMeshList, GRPolygonList and GRPointsList.

## Appendix A. classes GRCircle, GRConc, GRCylinder

Examples of using GRTriMesh.

### A.1 GRCircle

#### Purpose

Create a unit circle in the X-Z plane ( $y=0$ ), with the center at the origin.

#### Libraries

GROOP library (libgroop.a)  
Math library (libm.a)

#### Syntax

```
#include <GRCircle.h>
GRCircle circle;
GRCircle circle(up);
GRCircle circle(slices, start, degrees, up);

GRCircle circle;
int slices;
float start, degrees;
int up;
```

#### Description

Create a circle in the X-Z plane ( $y = 0$ ), and a radius of 1.0. The center is at the origin (0.0, 0.0, 0.0).

GROOP automatically converts the specification of the circle into a set of triangles (GRTriMesh). The number of triangles is, by default, set to 18. This can be overridden by specifying the number of slices.

Partial circles can be constructed by specifying the start angle, and the number of degrees.

The surface normal can also be specified.

#### Parameters

<b>slices</b>	the number of points around the perimeter of the circle to use when specifying the triangles used to approximate a circle for rendering.
<b>start</b>	the angle offset to use when specifying a partial circle.
<b>angle</b>	the amount of the circle to be rendered when specifying a partial circle.
<b>up</b>	the surface normal can be specified - 1 for up and -1 for down.

#### Implementation Specifics

Because most 3-D graphics packages do not have a routine for directly rendering shaded 3-D circles, GROOP converts the circle into a strip of triangles.

GRCircle inherits from GRTriMesh.

#### Related Information

See GRTriMesh, GRGeometricObject, GRrgb, GRConc and GRCylinder.

## A.2 GRCono

### Purpose

Create a unit cone defined around the y axis, and the tip of the cone at (0.0, 1.0, 0.0). The center of the base is at (0.0, 0.0, 0.0).

### Libraries

GROOP library (libgroop.a)  
Math library (libm.a)

### Syntax

```
#include <GRCono.h>
GRCono cone;
GRCono cone(slices);
GRCono cone(slices, start, degrees);

GRCono cone;
int slices;
float start, degrees;
```

### Description

Create a unit cone defined around the y axis, and the tip of the cone at (0.0, 1.0, 0.0). The center of the base is at (0.0, 0.0, 0.0), with a radius of 1.0.

GROOP automatically converts the specification of the cone into a set of triangles (GRTriMesh). The number of triangles is, by default, set to 18. This can be overridden by specifying the number of slices.

Partial cones can be constructed by specifying the start angle, and the number of degrees.

### Parameters

<b>slices</b>	the number of points around the perimeter of the cone to use when specifying the triangles used to approximate a cone for rendering.
<b>start</b>	the angle offset to use when specifying a partial cone.
<b>angle</b>	the amount of the cone to be rendered when specifying a partial cone.

### Implementation Specifics

Because most 3-D graphics packages do not have a routine for directly rendering shaded 3-D cones, GROOP converts the cone into a strip of triangles.

GRCono inherits from GRTriMesh.

### Related Information

See GRTriMesh, GRGeometricObject, GRrgb, GRCircle and GRCylinder.

## A.3 GRCylinder

### Purpose

Create a unit cylinder defined around the y axis, with the top at  $y = 0.5$  and bottom at  $y = -0.5$ .

### Libraries

GROOP library (libgroop.a)  
Math library (libm.a)

### Syntax

```
#include <GRCylinder.h>
GRCylinder cylinder;
GRCylinder cylinder(slices);
GRCylinder cylinder(slices, start, degrees);

GRCylinder cylinder;
int slices;
float start, degrees;
```

### Description

Create a unit cylinder defined around the y axis, with the top at  $y = 0.5$  and bottom at  $y = -0.5$ . The radius is 1.0.

GROOP automatically converts the specification of the cylinder into a set of triangles (GRTriMesh). The number of triangles is by default set to 18. This can be overridden by specifying the number of slices.

Partial cylinders can be constructed by specifying the start angle, and the number of degrees.

### Parameters

<b>slices</b>	the number of points around the perimeter of the cylinder to use when specifying the triangles used to approximate a cylinder for rendering.
<b>start</b>	the angle offset to use when specifying a partial cylinder.
<b>angle</b>	the amount of the cylinder to be rendered when specifying a partial cylinder.

### Implementation Specifics

Because most 3-D graphics packages do not have a routine for directly rendering shaded 3-D cylinders, GROOP converts the cylinder into a strip of triangles.

GRCylinder inherits from GRTriMesh.

### Related Information

See GRTriMesh, GRGeometricObject, GRrgb, GRCircle and GRCone.

## Appendix B. classes **GRCappedCone**, **GRCappedCylinder**

Examples of using GRComposite.

### B.1 **GRCappedCone**

#### **Purpose**

Create a unit cone with a covered base.

#### **Libraries**

GROOP library (libgroop.a)

#### **Syntax**

```
#include <GRCone.h>
GRCappedCone cone;
GRCappedCone cone(slices);

GRCappedCone cone;
int slices;
```

#### **Description**

Create a capped cone from the composition of a cone and circle (GRCone and GRCircle objects).

The number of triangles used to create the renderable object is optionally specified in the constructor, and is passed directly to the constructors for GRCone and GRCircle).

#### **Parameters**

**slices** the number of points around the perimeter of the cone and circle to use when specifying the triangles used to approximate a cone and circle for rendering.

#### **Implementation Specifics**

GRCappedCone inherits from GRComposite. The cone and circle are created from GRCone and GRCircle classes and added to the composite.

Partial capped cones are not supported.

#### **Related Information**

See GRGeometricObject, GRrgb, GRCone, GRCircle, GRCappedCylinder and GRComposite.

## B.2 GRCappedCylinder

### Purpose

Create a unit cylinder defined around the y axis, with the top at  $y = 0.5$  and bottom at  $y = -0.5$ , capped at both ends with a circle.

### Libraries

GROOP library (libgroop.a)

### Syntax

```
#include <GRCappedCylinder.h>
GRCappedCylinder cylinder;
GRCappedCylinder cylinder(slices);

GRCappedCylinder cylinder;
int slices;
```

### Description

Create a unit cylinder defined around the y axis, with the top at  $y = 0.5$  and bottom at  $y = -0.5$ . The radius is 1.0. Both ends of the cylinder are capped with circles.

GROOP automatically converts the specification of the capped cylinder into a set of triangles (GRTriMesh). The number of triangles is by default set to 18 for the cylinder and two circles (see GRCylinder and GRCircle). This can be overridden by specifying the number of slices.

### Parameters

**slices** the number of points around the perimeter of the cylinder to use when specifying the triangles used to approximate a cylinder and circle for rendering.

### Implementation Specifics

GRCappedCone inherits from GRComposite. The cone and circle are created and added to the composite.

Partial capped cones are not supported.

### Related Information

See GRGeometricObject, GRrgb, GRCylinder, GRCircle and GRCappedCone.

## Appendix C. class GRCube

An example of using GRPolygonList.

### C.1 GRCube

#### Purpose

Create a unit cube with the center at the origin and faces parallel to the planes X-Y, X-Z and Y-Z.

#### Libraries

GROOP library (libgroop.a)

#### Syntax

```
#include <GRCube.h>
GRCube cube;
```

#### Description

Create a unit cube with the center at the origin and faces parallel to the planes X-Y, X-Z and Y-Z.

#### Implementation Specifics

GRCube inherits from GRPolygonList (each face of the cube is a polygon).

#### Related Information

See GRGeometricObject and GRrgb, GRCircle, GRCone, GRCylinder, GRCappedCone, GRCappedCylinder, GRSphere and GRTorus.

## Appendix D. classes **GRTorus**, **GRSphere**, **GRBall**, **GRAxis**

Examples of using GRComposite.

### D.1 **GRTorus**

#### Purpose

Create a unit torus, radius of 1.0, ring radius of 0.5.

#### Libraries

GROOP library (libgroop.a)

Math library (libm.a)

#### Syntax

```
#include <GRTorus.h>
GRTorus   torus;
GRTorus   torus(radius, thickness, latitudes, longitudes);
GRTorus   torus(radius, thickness, latitudes, longitudes,
                  startLatAngle, forLatDegrees,
                  startLongAngle, forLongDegrees);

GRTorus   torus;
float     radius, thickness;
int       latitudes, longitudes;
float     startLatAngle, forLatDegrees;
float     startLongAngle, forLongDegrees;
```

#### Description

Create a torus or partial torus.

#### Parameters

<b>radius</b>	the distance from the origin to the center of the torus ring.
<b>thickness</b>	the radius of the torus ring.
<b>latitudes</b>	when converting the torus into triangles, this value specifies the number of faces from the top to the bottom of the torus.
<b>longitudes</b>	when converting the torus into triangles, this value specifies the number of faces around the ring.
<b>startLatAngle</b>	for a partial torus, this specifies the starting angle for the ring.
<b>forLatDegrees</b>	for a partial torus, this specifies the number of degrees of the torus ring to render.
<b>startLongAngle</b>	for a partial torus, this specifies the starting angle for rendering the body of the ring.
<b>forLongDegrees</b>	for a partial torus, this specifies the number of degrees of the torus ring body to render.

#### Implementation Specifics

Because most 3-D graphics packages do not have a routine for directly rendering a shaded 3-D torus, GROOP converts the torus into a list of triangle meshes.

GRTorus inherits from GRComposite. Each of the triangle meshes are added to the composite.

### **Related Information**

See GRGeometricObject, GRrgb, GRSphere, GRCone, GRCube, GRCircle, GRCappedCylinder, GRCappedCylinder and GRComposite.

## D.2 GRSphere

### Purpose

Create a unit sphere, radius of 1.0.

### Libraries

GROOP library (libgroop.a)

Math library (libm.a)

### Syntax

```
#include <GRSphere.h>
GRSphere sphere;
GRSphere sphere(latitudes, longitudes,
                startLatAngle, forLatDegrees,
                startLongAngle, forLongDegrees);

GRSphere sphere;
int latitudes, longitudes;
float startLatAngle, forLatDegrees;
float startLongAngle, forLongDegrees;
```

### Description

Create a sphere or partial sphere.

### Parameters

<b>latitudes</b>	when converting the sphere into triangles, this value specifies the number of faces from the top to the bottom of the sphere.
<b>longitudes</b>	when converting the sphere into triangles, this value specifies the number of faces around the sphere.
<b>startLatAngle</b>	for a partial sphere, this specifies the starting angle for the sphere.
<b>forLatDegrees</b>	for a partial sphere, this specifies the number of degrees of the sphere to render.
<b>startLongAngle</b>	for a partial sphere, this specifies the starting angle for rendering of the sphere.
<b>forLongDegrees</b>	for a partial sphere, this specifies the number of degrees of the sphere to render.

### Implementation Specifics

Because most 3-D graphics packages do not have a routine for directly rendering a shaded 3-D torus, GROOP converts the sphere into a set of triangle meshes.

GRSphere inherits from GRComposite. Each of the triangle meshes are added to the composite.

### Related Information

See GRGeometricObject, GRrgb, GRTorus, GRCone, GRCube, GRCircle, GRCappedCylinder, GRCappedCylinder and GRComposite.

## D.3 GRBall

### Purpose

Create a striped ball that can be scaled.

### Libraries

GROOP library (libgroop.a)

### Syntax

```
#include <GRBall.h>
GRBall  ball(scale);

GRBall  ball;
float   scale;
```

### Description

Create a striped ball (red with a yellow band) that is scaled.

### Parameters

**scale**                    scaling factor for the ball.

### Implementation Specifics

The ball is a GRComposite, composed of three partial sphere's, each part having its own color attributes.

This implementation does not allow for changing the size of the stripe, or the ball's colors.

### Related Information

See GRGeometricObject, GRrgb, GRSphere, GRTorus, GRConc, GRCube, GRCircle, GRCappedCylinder, GRCappedCylinder and GRComposite.

## D.4 GRAxis

### Purpose

Draw and label an X-Y-Z axis.

### Libraries

GROOP library (libgroop.a)

### Syntax

```
#include <GRAxis.h>
GRAxis axis(scale, axisWidth, fontName);
void axis.XAxisLabel(label);
void axis.YAxisLabel(label);
void axis.ZAxisLabel(label);

GRAxis axis;
float scale;
int axisWidth;
char* fontName;
char* label;
```

### Description

Create an X-Y-Z axis with text labels. Each axis is drawn from the origin to 1.0, and then scaled according to the scale value. The default labels are "X", "Y" and "Z". These labels can be changed using the appropriate member function.

### Parameters

<b>scale</b>	scaling factor for the axis.
<b>axisWidth</b>	value passed to GRLine to define the line width.
<b>fontName</b>	font name passed to GRText for labelling the axes.
<b>label</b>	the text used to label the axis.

### Implementation Specifics

GRAxis objects are derived from GRComposite objects. Axes are drawn using GRLine objects and labelled by using GRText objects.

### Related Information

See GRGeometricObject, GRrgb and GRComposite.