Quick Reference Sheet

Installation & Compilation

Installation preliminaries:
Linux: sudo apt-get install **subversion**
Mac: install **XCode**
Windows 10: install **MinGW 7.3** 64-bit, **TortoiseSVN**, and **Qt 5.12**
*Installation via the **installer** script:*
svn checkout https://svn.code.sf.net/p/glvertex/code/install
cd install; **./installer.sh** (on Windows run **installer.bat**)

*Building the programming template **qt_template.cpp**:*
CMake: cmake . && make
QtCreator: open qt_template.pro, press hammer button
XCode: run generate.sh script, open .xcodeproj

Creating a new project:
copy qt_template.cpp, qt_template.pro and CMakeLists.txt into a **new** directory and rename both qt_template.cpp and qt_template.pro

Terminal commands:
cd ls pwd mkdir cp mv rm top ping more less

Trouble shooting with the software OpenGL rasterizer:
export **LIBGL_ALWAYS_SOFTWARE**=1

Basics

Editable methods in qt_template.cpp:
C++ constructor: variable initialization
initializeOpenGL(): executed once
renderOpenGL(dt): executed once per rendered frame
dt is the time in seconds since the last rendered frame

```
// clear frame buffer
lglClearColor(0,0,0);
lglClear();

// render a diagonal line
lglBegin(LGL_LINES);
  lglVertex(-1,-1,0);
  lglVertex(1,1,0);
lglEnd();
```

Ctrl-q, ESC: quit
Ctrl-f: fullscreen mode

```
// projection setup
lglProjection(field of view,
              window aspect,
              near plane dist,
              far plane dist);

// viewing setup
lglView(vec3(eye point),
        vec3(lookat pos),
        vec3(up vector));

// modeling transformations
lglTranslate(vec3(vector))
lglRotate(degrees, vec3(axis))
lglScale(factor)
```

GLSLmath

$$M_P = \begin{pmatrix} \frac{2n}{w} & 0 & 0 & 0 \\ 0 & \frac{2n}{h} & 0 & 0 \\ 0 & 0 & -\frac{n+f}{f-n} & -2\frac{fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

Creating a 3D vector: **vec3** v(1,2,3);
Accessing vector components: double x = v**.x**;
Printing a vector: **std::cout** << "v = " << v << std::endl;

Getting the length of a vector: double l = **length**(v);
Calculating the dot product: double d = **dot**(v1, v2);
Calculating the cross product: vec3 c = **cross**(v1, v2);
Normalized direction vector: vec4 d = **normalize**(p2-p1);
Component swizzeling: a.**wzyx**(), b.**xy**(), ...

Identity matrix: **mat4** M;
Pretty-printing a matrix M: glslmath::**print**(M);
Matrix/Vector multiplication: v = **M*vec4(v)**;

Matrix transformations:
mat4::**translate**(vec3(vector))
mat4::**rotate**(degrees, vec3(axis));
mat4::**scale**(factor)

Model-View and Projection Matrix calculation:
mat4 **P** = mat4::**perspective**(fovy, aspect, near, far);
mat4 **V** = mat4::**lookat**(vec3(eye), vec3(lookat), vec3(up));
mat4 **M** = mat4::translate(0,0,-10) * mat4::rotate(90, 0,1,0);
mat4 MVP = **P*V*M**;
mat4 MVIT = **inverse**(**transpose**(V*M));

Geometry

Rendering a colored triangle:

```
// render triangle
lglBegin(LGL_TRIANGLES);
  lglColor(1,0,0);
  lglVertex(-0.5,-0.5,0);
  lglColor(0,1,0);
  lglVertex(0.5,-0.5,0);
  lglColor(0,0,1);
  lglVertex(0,0.5,0);
lglEnd();
```

Geometric primitives:
LGL_LINES, LGL_LINE_STRIP
LGL_TRIANGLES, LGL_TRIANGLE_STRIP
LGL_QUADS, LGL_QUAD_STRIP

Per-vertex attributes:
lglColor(): interpolated colors
lglNormal(): normals for lighting
lglTexCoord(): texture coordinates for texture mapping

*The built-in mouse **trackball** rotates the scene.*
Ctrl-w: enable wireframe mode = **lglPolygonMode(LGL_LINE)**

VBOs

*Pre-defined **unit-size** VBOs:*
lglCube, lglWireCube, lglBox
lglTet, lglPyramid, lglPrism
lglSphere, lglHemisphere, lglCylinder, lglHemiCylinder
lglDisc, lglHemiDisc, lglCone
lglRing, lglArc, lglTorus, lglHemiTorus
lglTeapot, lglCoordSys

VBO usage:

```
// declare vbo (as a member variable)
lglTeapot teapot;

// render vbo (in the renderOpenGL method)
lglRender(teapot);
```

VBO creation:

```
lglVBO vbo;
vbo.lglBegin(LGL_TRIANGLES);
  vbo.lglColor(1,0,0); vbo.lglVertex(-0.5,-0.5,0);
  vbo.lglColor(0,1,0); vbo.lglVertex(0.5,-0.5,0);
  vbo.lglColor(0,0,1); vbo.lglVertex(0,0.5,0);
vbo.lglEnd();
```

Loading a VBO from an OBJ file (in the app dir):
lglVBO *vbo = **lglLoadObj**("teapot.obj");

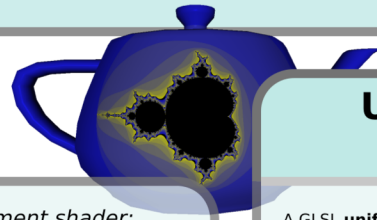
Lighting & Texturing

The specification of per-vertex normals (with **lglNormal**) automatically triggers **Blinn-Phong** shading with a single white head light.

The specification of per-vertex texture coordinates (with **lglTexCoord**) automatically triggers OpenGL legacy texture mapping. A texture object needs to be specified with **lglTexture2D**(). Texture objects are created with **lglCreateTexmap2D**() or **lglCreateMipmap2D**().

Loading an image file (in the app dir) into a texture object:
GLuint texid = **lglLoadQtTexture**("image.png");

Quick Reference Sheet



Shaders

The shortest combined vertex and fragment shader:

```
#version 120
attribute vec4 vertex_position;
uniform mat4 mvp;
void main()
{
  gl_Position = mvp * vertex_position;
}
---
#version 120
uniform vec4 color;
void main()
{
  gl_FragColor = color;
}
```

The above shader is called "**plain shader**".
The uniform model-view-projection matrix "**mvp**" is set automatically from preceding calls of lglProjection() and lglView().

Uniforms & Varyings

A GLSL **uniform** is a **global parameter** of the shader.

The uniforms of the currently active GLSL program are set via lglUniform*(). Those uniforms need to be specified after lglUseProgram():

lglUniform[i/f/fv]("name", value);

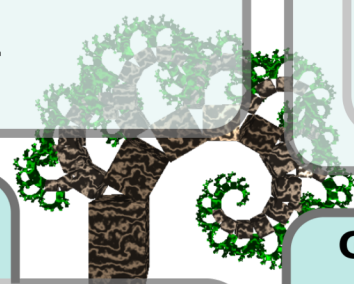
Suffix **i** is for integer, **f** for float and **fv** for float arrays (vectors and matrices).

Uniform samplers can be set with **lglSampler2D**(), which is just a convenience wrapper around lglUniformi() and lglTexture2D().

A GLSL **varying** is a **data channel** between the fragment and the vertex shader. On both sides it needs to be declared exactly the same:

```
// vertex shader
varying vec4 vary;
...
void main()
{
  vary = ...;
  gl_Position = ...;
}
```

```
// fragment shader
varying vec4 vary;
...
void main()
{
  vec4 v = vary;
  gl_FragColor = v;
}
```



GLSL

The GLSL program must comply to the following rules:

- Vertices are passed in the attribute "**vertex_position**" (vec4).
- Colors are passed in the attribute "**vertex_color**" (vec4).
- Normals are passed in the attribute "**vertex_normal**" (vec3).
- Texture coordinates are passed in the attribute "**vertex_texcoord**" (vec4).
- The vertex shader may use the model-view-projection matrix "**mvp**" and transform the vertices with that matrix (uniform mat4 mvp).
- The fragment shader may use the actual color (uniform vec4 **color**).
- If normals were specified, the vertex shader may use the model-view matrix "**mv**" resp. the inverse transpose model-view matrix "**mvit**" to transform the vertex normals (uniform mat4).
- The vertex shader is required to write "**gl_Position**" (vec4).
- The fragment shader is required to write "**gl_FragColor**" (vec4).

Compiling a shader from inlined source:
GLuint program = **lglCompileGLSLProgram**("#version 120\n ...");

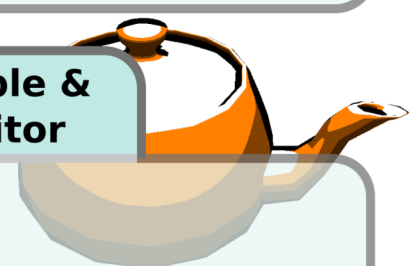
Activating a compiled shader:
lglUseProgram(program);

Loading a model-view matrix M into the built-in uniform "mv" (resp. "mvit):
lglModelView(M);

Getting the trackball manipulator matrix:
mat4 M = **lglGetManip**();

Deleting a compiled shader:
lglDeleteGLSLProgram(program);

GLSL Example & Shader Editor



Fogging shader:

```
static const char shader[] =
  "#version 120\n"
  "attribute vec4 vertex_position;\n"
  "attribute vec4 vertex_color;\n"
  "uniform mat4 mvp;\n"
  "varying vec4 frag_color;\n"
  "vec4 fvertex() {return(mvp * vertex_position);}\n"
  "void main()\n"
  "{\n"
  "  frag_color = vertex_color;\n"
  "  gl_Position = fvertex();\n"
  "}\n"
  "---\n"
  "#version 120\n"
  "uniform float density;\n"
  "varying vec4 frag_color;\n"
  "void main()\n"
  "{\n"
  "  float z = 1.0f/gl_FragCoord.w;\n"
  "  float f = 1.0f-exp(-density*z*z);\n"
  "  gl_FragColor = (1.0f-f)*frag_color + f*vec4(1);\n"
  "}\n";

GLuint program = lglCompileGLSLProgram(shader);
create_lgl_Qt_ShaderEditor("shader", &program);
lglUseProgram(program);
lglUniformf("density", 0.1f);
```

Programming API

C++

*API functions as specified by **OpenGL 1.2**:*
lglBegin, **lglEnd**
lglVertex, lglColor, lglNormal, lglTexCoord
Matrix and modeling functions:
lglLoadIdentity, lglMatrixMode
lglLoadMatrix, lglMultMatrix
lglScale, **lglTranslate**, **lglRotate**
lglOrtho, lglFrustum, lglPerspective, lglLookAt
lglPushMatrix, lglPopMatrix
Miscellaneous functions:
lglClear, **lglClearColor**, lglViewport
lglLight, lglClipPlane, lglFog
lglLineWidth, lglPolygonMode,
lglDepthTest, lglBackFaceCulling
lglGetError

Extended convenience functions:
lglProjection, **lglView**, **lglModelView**, lglTexture
lglLoadObj, **lglRender**
Texturing functions:
lglLoadQtTexture, **lglTexture2D**,
lglCreateTexmap2D, lglCreateMipmap2D
GLSL functions:
lglCompileGLSLProgram, **lglUseProgram**, lglDeleteGLSLProgram
lglLoadGLSLProgram, lglPlainGLSLProgram
lglGetManip, lglGetInverseTransposeManip
lglUniformi, **lglUniformf**, **lglUniformfv**
lglSampler2D

Please note that the above command overview represents only a subset of the API.
For more details see the quick reference documentation (**QUICKREF.txt**).