APPENDIX D

SYNOPSIS OF OPENGL FUNCTIONS

D.1 INITIALIZATION AND WINDOW FUNCTIONS

```
void glutInit(int *argc, char **argv)
```

initializes GLUT. The arguments from `main` are passed in and can be used by the application.

```
void glewInit(void)
```

initializes GLEW if used by application.

```
int glutCreateWindow(char *title)
```

creates a window on the display. The string `title` can be used to label the window. The return value provides a reference to the window that can be used when there are multiple windows.

```
void glutInitDisplayMode(unsigned int mode)
```

requests a display with the properties in `mode`. The value of `mode` is determined by the logical OR of options including the color model (`GLUT_RGB`, `GLUT_INDEX`) and buffering (`GLUT_SINGLE`, `GLUT_DOUBLE`).

```
void glutInitWindowSize(int width, int height)
```

specifies the initial height and width of the window in pixels.

```
void glutInitWindowPosition(int x, int y)
```

specifies the initial position of the top-left corner of the window in pixels.

```
void glViewport(int x, int y, GLsizei width, GLsizei height)
```

specifies a `width` × `height` viewport in pixels whose lower-left corner is at (`x`, `y`) measured from the origin of the window.

```
void glutMainLoop()
```

causes the program to enter an event-processing loop. It should be the last statement in `main`.

```
void glutDisplayFunc(void (*func)(void))
```

registers the display function `func` that is executed when the window needs to be redrawn.

```
void glutPostRedisplay()
```

requests that the display callback be executed after the current callback returns.

```
void glutSwapBuffers()
```

swaps the front and back buffers.

```
void glFlush()
```

forces any buffered OpenGL commands to execute.

```
void glutSetWindow(int id)
```

sets the current window to the window with identifier `id`.

```
void glutContextVersion(int major_version, int minor_version)
```

sets the desired context, e.g., `glutContextVersion(3, 1)` for OpenGL 3.1. Only available in freeglut.

```
void glutContextProfile(init profile)
```

sets the desired context to either `GLUT_CORE_PROFILE` or `GLUT_COMPATIBILITY_PROFILE`. Compatibility profile allows backward compatibility. Only available with freeglut.

D.2 VERTEX BUFFER OBJECTS

```
void glGenVertexArrays(GLsizei n, GLuint *array)
```

generates `n` unused identifiers for vertex array objects in `array`.

```
void glBindVertexArray(GLuint id)
```

creates a new vertex array object with identifier `id`. Subsequent calls with the same identifier make that the active array.

```
void glGenBuffers(GLsizei n, GLuint *buffer)
```

generates `n` unused identifiers for buffer objects in `buffer`.

```
void glBindBuffer(GLenum target, GLint id)
```

creates a new buffer object with identifier `id`. Subsequent calls with the same identifier make that the active buffer object. The type of buffer object is given by `target`. Types include `GL_ARRAY_BUFFER` for vertex attribute data.

```
void glBufferData(GLenum target, GLsizeiptr size, const GLvoid *data,
                  GLenum usage)
```

allocates `size` units of server memory for vertex array objects of type `target` pointed to by `data`. Types include `GL_ARRAY_BUFFER`. The `usage` parameter specifies how the data will be read and includes `GL_STATIC_DRAW` and `GL_DYNAMIC_DRAW`.

```
void glBufferSubData(GLenum target, GLintiptr offset, GLsizeiptr size,
                     const GLvoid *data)
```

updates `size` bytes starting at `offset` in the current buffer object with data of type `target` starting at `data`.

```
void glVertexAttrib[1234][sfd](GLunit index, TYPE data);
void glVertexAttrib[1234][sfd]v(GLunit index, TYPE *data);
```

specifies `values` for vertex attributes with the given `index`.

```
void glVertexAttribPointer(GLuint index, GLint size, GLenum type,
   GLboolean normalized, GLsizei stride, const GLvoid* data)
```

points to `data` where vertex data of `size` components corresponding to `index` are stored. Data are one of the standard types such as `GL_INT` and `GL_FLOAT`. If `normalized` is set to `GL_TRUE`, the data will be normalized when stored. If `stride` is set to 0, the data are assumed to be contiguous.

```
void glEnableVertexAttribArray(GLuint index)
```

enables the vertex array with identifier `index`.

```
void glDrawArrays(GLenum mode, GLint first,  GLsizei count)
```

creates `count` elements of the standard OpenGL types `mode`, such as `GL_TRIANGLES` or `GL_LINES` starting at `first`.

D.3 INTERACTION

```
void glutMouseFunc(void *f(int button, int state, int x, int y))
```

registers the mouse callback function `f`. The callback function returns the button (`GLUT_LEFT_BUTTON`, `GLUT_MIDDLE_BUTTON`, `GLUT_RIGHT_BUTTON`), the state of the button after the event (`GLUT_UP`, `GLUT_DOWN`), and the position of the mouse relative to the top-left corner of the window.

```
void glutReshapeFunc(void *f(int width, int height))
```

registers the reshape callback function `f`. The callback returns the height and width of the new window. The reshape callback invokes a display callback.

```
void glutKeyboardFunc(void *f(char key, int width, int height))
```

registers the keyboard callback function `f`. The callback function returns the ASCII code of the key pressed and the position of the mouse.

```
void glutIdleFunc(void (*f)(void))
```

registers the display callback function `f` that is executed whenever there are no other events to be handled.

```
int glutCreateMenu(void (*f)(int value))
```

returns an identifier for a top-level menu and registers the callback function `f` that returns an integer `value` corresponding to the menu entry selected.

```
void glutSetMenu(int id)
```

sets the current menu to the menu with identifier `id`.

```
void glutAddMenuEntry(char *name, int value)
```

adds an entry with the string `name` displayed to the current menu. `value` is returned to the menu callback when the entry is selected.

```
void glutAttachMenu(int button)
```

attaches the current menu to the specified mouse `button`.

```
void glutAddSubMenu(char *name, int menu)
```

adds a submenu entry `name` to the current menu. The value of `menu` is the identifier returned when the submenu was created.

```
void glutTimerFunc(int delay, void (*f)(int v), int value)
```

registers the timer callback function `f` and delays the event loop by `delay` milliseconds. After the timer counts down, `f` is executed with the parameter v. `value` is available to `f`.

```
void glutMotionFunc(void (*f)(int x, int y))
```

registers the motion callback function `f`. The position of the mouse is returned by the callback when the mouse is moved at with least one of the mouse buttons pressed.

```
void glutPassiveMotionFunc(void (*f)(int x, int y))
```

registers the motion callback function `f`. The position of the mouse is returned by the callback when the mouse is moved.

D.4 SETTING ATTRIBUTES AND ENABLING FEATURES

```
void glEnable(GLenum feature)
```

enables an OpenGL `feature`. Features that can be enabled include `GL_DEPTH_TEST`, `GL_TEXTURE_1D`, `GL_TEXTURE_2D`, `GL_TEXTURE_3D`, `GL_LINE_SMOOTH`, `GL_POLYGON_SMOOTH`, `GL_POINT_SMOOTH`, `GL_BLEND`.

```
void glDisable(GLenum feature)
```

disables an OpenGL `feature`.

```
void glPolygonMode(glEnum faces, glEnum mode)
```

sets the desired mode for polygon rendering the faces (`GL_FRONT_AND_BACK`). `mode` can be `GL_POINTS`, `GL_LINES`, or `GL_FILL`.

```
void glClearColor(GLclampf r, GLclampf g, GLclampf b, GLclampf a)
```

sets the present RGBA clear color used when clearing the color buffer. Variables of type `GLclampf` are floating-point numbers between 0.0 and 1.0.

```
void glPointSize(GLfloat size)
```

sets the point size attribute in pixels.

```
void glPolygonOffset(GLfloat factor, GLfloat units)
```

offsets polygon depths by a linear combination of `factor` and `units`. The multiplicative constants in the computation depend on the slope of the polygon and the precision of the depth values.

```
glDepthMask(GLboolean flag)
```

sets `flag` to make the depth buffer read-only (`GL_FALSE`) or writable (`GL_TRUE`).

```
void glBlendFunc(GLenum source, GLenum destination)
```

sets the `source` and `destination` blending factors. Options include `GL_ONE`, `GL_ZERO`, `GL_SRC_COLOR`, `GL_SRC_ALPHA`, `GL_ONE_MINUS_SRC_COLOR`, `GL_ONE_MINUS_SRC_ALPHA`, `GL_DST_COLOR`, `GL_ONE_MINUS_DST_COLOR`, `GL_DST_ALPHA`, `GL_ONE_MINUS_DST_ALPHA`.

D.5 TEXTURE AND IMAGE FUNCTIONS

```
glTexImage2D[ui us f]v(GLenum target, GLint level, GLint iformat,
    GLsizei width, GLsizei height, GLint border, GLenum format,
    GLenum type, GLvoid *texels)
```

sets up a two-dimensional texture of `height` × `width` `texels` of `type` and `format`. The array `texels` is of format `iformat`. A `border` of 0 or 1 texels can be specified.

```
glTexParameter[if](GLenum target, GLenum param, TYPE value)
glTexParameter[if]v(GLenum target, GLenum param, TYPE *value)
```

sets the texture parameter `param` to `value` for texture of type `target` (`GL_TEXTURE_1D`, `GL_TEXTURE_2D`, or `GL_TEXTURE_3D`).

```
glGenTextures(GLsizei n, GLuint name)
```

returns in `name` the first integer of `n` unused integer for texture-object identifiers.

```
glBindTexture(GLenum target, GLuint name)
```

binds `name` to texture of type `target` (`GL_TEXTURE_1D`, `GL_TEXTURE_2D`, `GL_TEXTURE_3D`, `GL_TEXTURE_CUBE_MAP`).

```
glDeleteTextures(GLsizei n, GLuint *namearray)
```

deletes `n` texture objects from the array `namearray` that holds texture-object names.

D.6 STATE AND BUFFER MANIPULATION

```
void glDrawBuffer(GLenum buffer)
```

selects the color buffer `buffer` for rendering.

```
void glLogicOp(GLenum op)
```

selects one of the 16 logical writing modes if the feature `GL_COLOR_LOGIC_OP` is enabled. Modes include replacement (`GL_COPY`), the default, and exclusive or (`GL_XOR`).

```
glReadPixels(GLint x, GLint y, GLsizei width, GLsizei height,
          GLenum format, GLenum type, GLvoid *image)
```

reads a `width` × `height` rectangle of pixels from the present read buffer starting at `x`, `y` into the array `image`. The pixels are in the specified `format` in the read buffer and written as the specified data `type`.

```
glPixelStore[if](GLenum param, TYPE value)
```

sets the pixel store parameter `param` to `value`. Parameters include `GL_UNPACK_SWAP_BYTES`, `GL_PACK_SWAP_BYTES`, `GL_PACK_ALIGNMENT`, `GL_UNPACK_ALIGNMENT`.

D.7 QUERY FUNCTIONS

```
void glGetBooleanv(GLenum name, GLboolean *param)
void glGetIntegerv(GLenum name, GLinteger *param)
void glGetFloatv(GLenum name, GLfloat *param)
void glGetDoublev(GLenum name, GLdouble *param)
void glGetPointerv(GLenum name, GLvoid **param)
```

writes the present value of the parameter `name` into `param`.

```
int glutGet(GLenum state)
```

returns the current value of a GLUT state variable such as `GLUT_WINDOW_WIDTH`, `GLUT_WINDOW_HEIGHT`, `GLUT_ELAPSED_TIME`.

D.8 GLSL FUNCTIONS

```
GLuint glCreateProgram()
```

creates an empty program object and returns an identifier for it.

```
GLuint glCreateShader(GLenum type)
```

creates an empty shader object of `type` `GL_VERTEX_SHADER` or `GL_FRAGMENT_SHADER` and returns an identifier for it.

```
void glShaderSource(GLuint shader, GLsizei nstrings, const GLchar
     **strings, const GLint *lengths)
```

identifies the source code for `shader` as coming from an array of `nstrings` `strings` of `lengths` characters. If the shader is a single null-terminated string, then `nstrings` is 1 and `lengths` is `NULL`.

```
void glCompileShader(GLuint shader)
```

compiles shader object `shader`.

```
void glAttachShader(GLunit program, GLuint shader)
```

attaches shader object `shader` to program object `program`.

```
void glLinkProgram(GLuint program)
```

links together the application and shaders in program object `program`.

```
void glUseProgram(GLuint program)
```

makes `program` the active program object.

```
GLint glGetAttribLocation(GLuint program, const GLchar *name)
```

returns the index of the attribute `name` from the linked program object `name`.

```
void glVertexAttrib[1234][sfd](GLuint index, TYPE value1,
    TYPE value2,...)
void glVertexAttrib[123][sfd]v(GLuint index, TYPE *value)
```

specifies the `value` of the vertex attribute with the specified `index`.

```
GLint glGetUniformLocation(GLuint program, const GLchar *name)
```

returns the index of uniform variable `name` from the linked program object `program`.

```
void glUniform1234[if](GLint index, TYPE value)
void glUniform1234[if]v(GLint index, GLsizei num, TYPE value)
void glUniformMatrix[234]f(GLint index, GLsizei num,
     GLboolean transpose, const GLfloat *value)
```

sets the `value` of a uniform variable, array, or matrix with the specified `index`. For the array and matrix, `num` is the number of elements to be changed.

```
void glGetProgram(GLuint program, GLenum pname, GLinit *param)
```

returns in `param` the value of parameter `pname` for program object `program`. Parameters include link status `GL_LINK_STATUS`, which returns `GL_TRUE` or `GL_FALSE`, and `GL_INFO_LOG_LENGTH`, which returns the number of characters in the information log.

```
void glGetShaderiv(GLuint shader, GLenum pname, GLint *param)
```

returns in `param` the value of parameter `pname` for shader object `shader`. Parameters include compile status `GL_COMPILE_STATUS`, which returns `GL_TRUE` or `GL_FALSE`, and `GL_INFO_LOG_LENGTH`, which returns the number of characters in the information log.

```
void getProgramInfoLog(GLuint program, GLsizei maxL, GLsizei *len,
    GLchar *infoLog)
```

returns the info log string for program object `program` into the array `infoLog` of lenth `maxL` and the length of the string in `len`.

```
void getShaderInfoLog(GLuint program, GLsizei maxL, GLsizei *len,
    GLchar *infoLog)
```

returns the info log string for shader object `program` into the array `infoLog` of length `maxL` and the length of the string in `len`.