

P149064 IGP Graphics Robert McIntyre

by Robert McIntyre

FILE	2783_ROBERT_MCINTYRE_P149064_IGP_GRAPHICS_ROBERT_MCINTYRE_110623_1686644464.DOCX (7.97M)		
TIME SUBMITTED	12-APR-2019 03:28PM (UTC+0100)	WORD COUNT	8324
SUBMISSION ID	104684603	CHARACTER COUNT	64033

Interactive Graphics Programming

Animated Fly-through Scene

Robert McIntyre

M6X0100

Gordon Dickers

Contents

Introduction	3
What is OpenGL?	3
What is Blender?.....	3
Description of the scene	3
Structure and Design of the code	5
Objects in blender.....	5
User Guide	7
References	9
Appendices.....	10
Glut.h	10
objectloader.h.....	23
Main.cpp	25
objloader.cpp	43

Introduction

A art display scene has been created in c using OpenGL to demonstrate the use of 3D structures and animation and the use of textures. The aim was to learn about object space, world space and a camera view. When you want to move around, the space (world space) stays static and the camera view moves.

What is OpenGL?

OpenGL is an open source graphic library for rendering 2D and 3D graphics. OpenGL can be used on multiple platforms/ operating systems (Windows, Linux and MacOS). When using OpenGL in the C/C++ a library is needed. In this program the GLUT libraries were used. The GLUT libraries are an established version of OpenGL. There is a newer version that can be used which is called GLFW (Graphics library framework). [1]

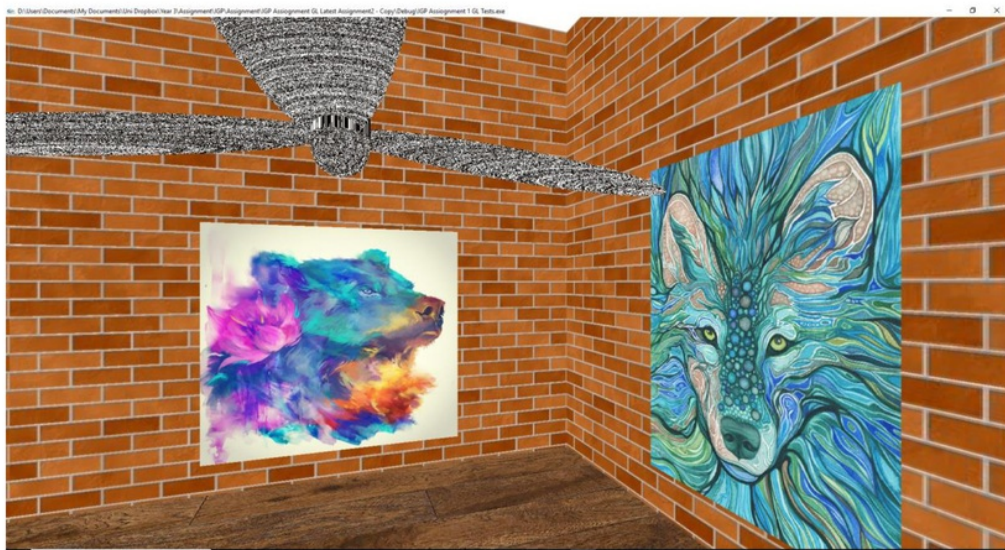
What is Blender?

Blender is also an open source program. This is used for 3d modelling. In this assignment blender was used to create the 3D models of the apple, table and fish and ceiling fan. After they were created the objects were then triangulated before they were exported. In blender the triangulate modifier turns the mesh from quads into triangles so that it can be imported into the C program. Triangulation is the process where the mesh is converted. [2] [3]

Description of the scene

The scene created is an art scene with two rooms. One with pictures and another with sculptures. The user can use a keyboard to move around the scene. The rooms were created using an OpenGL program. When the program first runs it will spawn you into the room with paintings on the walls. To the left is another room which contains models made in blender. One is a static fish the other is a rotating apple. Each of the rooms has Textured walls and floor and a rotating ceiling fan. There are light sources in the rooms set to maximum. In the room with sculptures each sculpture is set on top of a table. Colour and texture needs to be added to all the models and the ceiling fans. The texture on the wall behind the fish sculpture has been applied in a different direction. There are two open door ways between the two rooms.

Room with pictures and rotating ceiling fan



Room with sculptures, tables and ceiling fan



Doorways between rooms

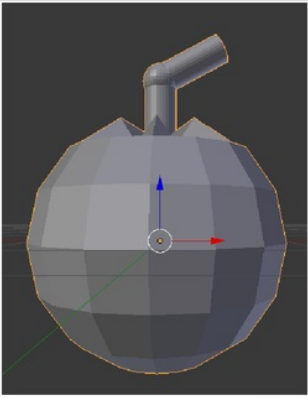
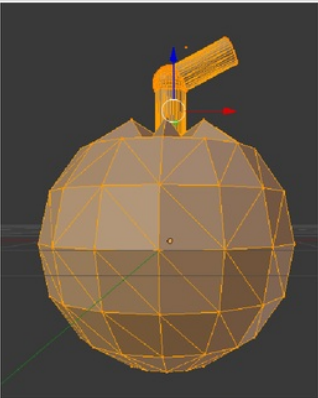
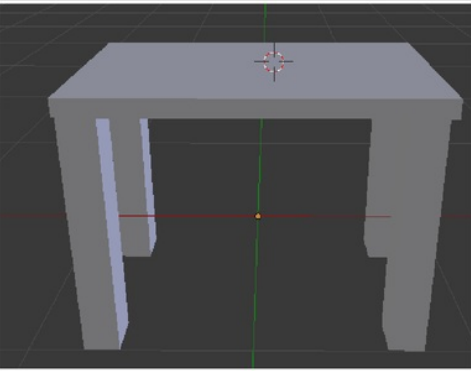
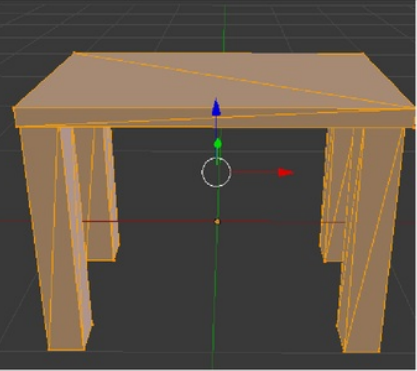
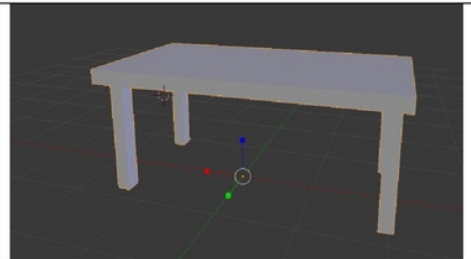
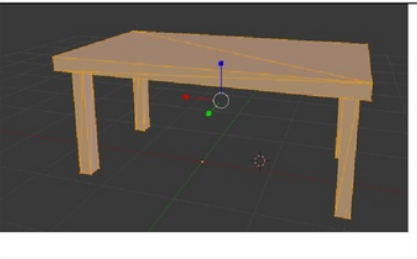
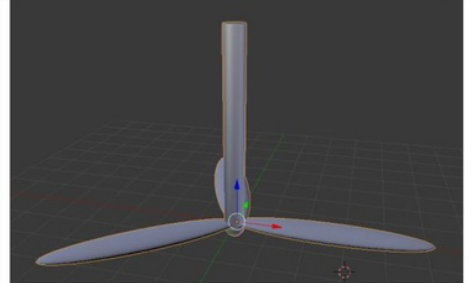
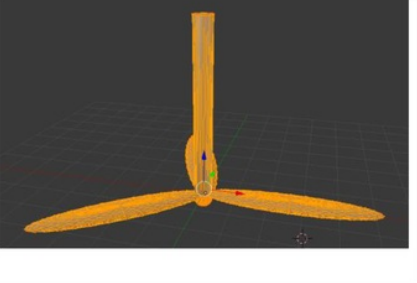


Structure and Design of the code

The project was created starting from the object loader code example found on Moodle. The project contains glut32.dll, glut32.lib and glut.h which are required for OpenGL to run. 6-lit scene with textures and animation example in moodle was used for the lighting and importing of bit maps. The images for the picture and the textures were sourced from artwork on the internet [4][5][6][7][8]. The 3D objects were created in blender before being imported into the program.

Objects in blender

	Model	Triangulated
Fish		

Apple		
Table		
Table Version 2		
Fan Object		

There are four different models that are loaded in the scene, Fish, Apple, Table and Fan. There were two designs of the table with the second version being chosen for the scene.

The objects are spread across the second room so when the user moves around the room, they can look at the objects around the room. Each room has a rotating ceiling fan and the table has been imported twice so that the sculptures can be placed on something.

User Guide

Installation of program

All the files required for the installation are found in the directory Assignment exe file. To run the program, click the exe file IGP Assignment 1 GL.exe

The program loads with the camera view in the picture room.

To navigate through the space the keyboard is used. All the keys are used without the CAPSLOCK as follows

w – to move forwards

s – to move backwards

a – to turn left

d – to turn right

spacebar – to move up

b – to move down

k – to rotate left

l – to rotate right

Whilst the appropriate key is pressed the camera will continue to move in that direction. Once the key is released the camera will stop.

Testing

Test No	Test Description	Expected Outcome	Actual Outcome	Pass / Fail
1	Test to see if the wall can be loaded in OpenGL	The Walls loads in the world space	The walls loads in the world space. This is the same for all the walls in the scene.	Pass
2	Test to see if the apple can load in to the world space	Apple loads in the world space with the correct translation	The apple loads with the correct translation	Pass
3	Testing to see if the apple rotates	The apple should rotate frequently	The apple rotate in a circle	Pass
4	Test to see if the table can load in to the world space	Table loads in the world space with the correct translation	The table loads with the correct translation	Pass

5	Test to see if the fish can load in to the world space	Fish loads in the world space with the correct translation	The fish loads with the correct translation	Pass
6	Test to see if the fan can load in to the world space	Fan loads in the world space with the correct translation	The fan loads with the correct translation	Pass
7	Testing to see if the fan rotates in the ceiling	The fan should rotate frequently	The fans rotate in a circle	Pass
8	Test to see if a .bmp files loads the texture on to the panel	The texture should be loaded on the plane	The texture is loaded on the planes on all walls, floor and pictures	Pass
9	Test to see if the texture loads on the objects	The texture should load on the object	The textures are not loading on the object as the texture looks strange.	Fail
10	Test for different lighting sources	The lights source should brighten up the scene	The light source is working as it is lighting up the whole two rooms	Pass
11	Strafe left and right	Should be able to move left and right depending what direction the camera is facing	This did not work as this went diagonally	Fail
12	Tilting up and down	The user should be able to tilt the camera up and down when pressing the k and l	This is tilting on the Z axis instead of the Y axis	Fail

References

- [1] D. Hearn, P. Baker, and W. R. Carithers, *Computer graphics with Open GL*. .
- [2] M. J. Donahoo, K. L. Calvert, and M. J. Donahoo, *TCP/IP sockets in C : practical guide for programmers*. Morgan Kaufmann, 2001.
- [3] “About — blender.org.” [Online]. Available: <https://www.blender.org/about/>. [Accessed: 01-Apr-2019].
- [4] https://static.boredpanda.com/blog/wp-content/uploads/2017/02/IMG_20170205_220039_508-58a0129a23ada__880.jpg
- [5] <https://static1.squarespace.com/static/580862acebbd1a723c82e81c/t/5c7ad73aee6eb044cf09f23b/1550778278585/?format=500w>
- [6] https://images.discerningassets.com/image/upload/q_auto/c_fit,h_600,w_600/v1486874136/GardenoftheWild-LION_zak45y.jpg
- [7] <http://www.americantrustflooring.com/wp-content/uploads/Wood-Flooring-1-768x512.jpg>
- [8] https://static6.depositphotos.com/1086305/578/i/950/depositphotos_5786515-stock-photo-urban-background-red-brick-wall.jpg

Appendices

Glut.h

```
#ifndef __glut_h__
#define __glut_h__

/* Copyright (c) Mark J. Kilgard, 1994, 1995, 1996, 1998. */

/* This program is freely distributable without licensing fees and is
   provided without guarantee or warranty expressed or implied. This
   program is -not- in the public domain. */

#if defined(_WIN32)

/* GLUT 3.7 now tries to avoid including <windows.h>
   to avoid name space pollution, but Win32's <GL/gl.h>
   needs APIENTRY and WINGDIAPI defined properly. */

/* if 0
   /* This would put tons of macros and crap in our clean name space. */
# define WIN32_LEAN_AND_MEAN
# include <windows.h>
# else
   /* XXX This is from Win32's <windef.h> */
# ifndef APIENTRY
# define GLUT_APIENTRY_DEFINED
# if (_MSC_VER >= 800) || defined(_STDCALL_SUPPORTED) || defined(__BORLANDC__) ||
defined(__LCC__)
# define APIENTRY __stdcall
# else
# define APIENTRY
# endif
# endif
# endif
/* XXX This is from Win32's <winnt.h> */
# ifndef CALLBACK
# if (defined(_M_MRX000) || defined(_M_IX86) || defined(_M_ALPHA) ||
defined(_M_PPC)) && !defined(MIDL_PASS) || defined(__LCC__)
# define CALLBACK __stdcall
# else
# define CALLBACK
# endif
# endif
/* XXX Hack for lcc compiler. It doesn't support __declspec(dllimport), just
__stdcall. */
# if defined(__LCC__)
# undef WINGDIAPI
# define WINGDIAPI __stdcall
# else
   /* XXX This is from Win32's <wingdi.h> and <winnt.h> */
# ifndef WINGDIAPI
# define GLUT_WINGDIAPI_DEFINED
# define WINGDIAPI __declspec(dllimport)
# endif
# endif
/* XXX This is from Win32's <ctype.h> */
# ifndef _WCHAR_T_DEFINED
typedef unsigned short wchar_t;
# define _WCHAR_T_DEFINED
# endif
# endif

/* To disable automatic library usage for GLUT, define GLUT_NO_LIB_PRAGMA
   in your compile preprocessor options. */
```

```

# if !defined(GLUT_BUILDING_LIB) && !defined(GLUT_NO_LIB_PRAGMA)
# pragma comment (lib, "winmm.lib") /* link with Windows MultiMedia lib */
/* To enable automatic SGI OpenGL for Windows library usage for GLUT,
define GLUT_USE_SGI_OPENGL in your compile preprocessor options. */
# ifdef GLUT_USE_SGI_OPENGL
# pragma comment (lib, "opengl.lib") /* link with SGI OpenGL for Windows lib */
# pragma comment (lib, "glu.lib") /* link with SGI OpenGL Utility lib */
# pragma comment (lib, "glut.lib") /* link with Win32 GLUT for SGI OpenGL lib */
# else
# pragma comment (lib, "opengl32.lib") /* link with Microsoft OpenGL lib */
# pragma comment (lib, "glu32.lib") /* link with Microsoft OpenGL Utility lib */
# pragma comment (lib, "glut32.lib") /* link with Win32 GLUT lib */
# endif
# endif

/* To disable supression of annoying warnings about floats being promoted
to doubles, define GLUT_NO_WARNING_DISABLE in your compile preprocessor
options. */
# ifndef GLUT_NO_WARNING_DISABLE
# pragma warning (disable:4244) /* Disable bogus VC++ 4.2 conversion warnings. */
# pragma warning (disable:4305) /* VC++ 5.0 version of above warning. */
# endif

/* Win32 has an annoying issue where there are multiple C run-time
libraries (CRTs). If the executable is linked with a different CRT
from the GLUT DLL, the GLUT DLL will not share the same CRT static
data seen by the executable. In particular, atexit callbacks registered
in the executable will not be called if GLUT calls its (different)
exit routine). GLUT is typically built with the
"/MD" option (the CRT with multithreading DLL support), but the Visual
C++ linker default is "/ML" (the single threaded CRT).

One workaround to this issue is requiring users to always link with
the same CRT as GLUT is compiled with. That requires users supply a
non-standard option. GLUT 3.7 has its own built-in workaround where
the executable's "exit" function pointer is covertly passed to GLUT.
GLUT then calls the executable's exit function pointer to ensure that
any "atexit" calls registered by the application are called if GLUT
needs to exit.

Note that the __glut*WithExit routines should NEVER be called directly.
To avoid the atexit workaround, #define GLUT_DISABLE_ATEXIT_HACK. */

/* XXX This is from Win32's <process.h> */
# if !defined(_MSC_VER) && !defined(__cdecl)
/* Define __cdecl for non-Microsoft compilers. */
# define __cdecl
# define GLUT_DEFINED___CDECL
# endif
# ifndef _CRTIMP
# ifdef _NTSDK
/* Definition compatible with NT SDK */
# define _CRTIMP
# else
/* Current definition */
# ifdef _DLL
# define _CRTIMP __declspec(dllimport)
# else
# define _CRTIMP
# endif
# endif
# endif

```

```

# define GLUT_DEFINED__CRTIMP
# endif

/* GLUT API entry point declarations for Win32. */
# ifdef GLUT_BUILDING_LIB
# define GLUTAPI __declspec(dllexport)
# else
# ifdef _DLL
# define GLUTAPI __declspec(dllimport)
# else
# define GLUTAPI extern
# endif
# endif

/* GLUT callback calling convention for Win32. */
# define GLUTCALLBACK __cdecl

#endif /* _WIN32 */

#include <GL/gl.h>
#include <GL/glu.h>

#ifdef __cplusplus
extern "C" {
#endif

#ifdef _WIN32
# if defined(GLUT_BUILDING_LIB)
# else
extern _CRTIMP void __cdecl exit(int);
# endif
#else
/* non-Win32 case. */
/* Define APIENTRY and CALLBACK to nothing if we aren't on Win32. */
# define APIENTRY
# define GLUT_APIENTRY_DEFINED
# define CALLBACK
/* Define GLUTAPI and GLUTCALLBACK as below if we aren't on Win32. */
# define GLUTAPI extern
# define GLUTCALLBACK
/* Prototype exit for the non-Win32 case (see above). */
extern void exit(int);
#endif

/**
GLUT API revision history:

GLUT_API_VERSION is updated to reflect incompatible GLUT
API changes (interface changes, semantic changes, deletions,
or additions).

GLUT_API_VERSION=1 First public release of GLUT. 11/29/94

GLUT_API_VERSION=2 Added support for OpenGL/GLX multisampling,
extension. Supports new input devices like tablet, dial and button
box, and Spaceball. Easy to query OpenGL extensions.

GLUT_API_VERSION=3 glutMenuStatus added.

GLUT_API_VERSION=4 glutInitDisplayString, glutWarpPointer,
glutBitmapLength, glutStrokeLength, glutWindowStatusFunc, dynamic
video resize subAPI, glutPostWindowRedisplay, glutKeyboardUpFunc,
glutSpecialUpFunc, glutIgnoreKeyRepeat, glutSetKeyRepeat,

```

```

    glutJoystickFunc, glutForceJoystickFunc (NOT FINALIZED!).
**/
#ifndef GLUT_API_VERSION /* allow this to be overridden */
#define GLUT_API_VERSION 3
#endif

/**
    GLUT implementation revision history:

    GLUT_XLIB_IMPLEMENTATION is updated to reflect both GLUT
    API revisions and implementation revisions (ie, bug fixes).

    GLUT_XLIB_IMPLEMENTATION=1 mjk's first public release of
    GLUT Xlib-based implementation. 11/29/94

    GLUT_XLIB_IMPLEMENTATION=2 mjk's second public release of
    GLUT Xlib-based implementation providing GLUT version 2
    interfaces.

    GLUT_XLIB_IMPLEMENTATION=3 mjk's GLUT 2.2 images. 4/17/95

    GLUT_XLIB_IMPLEMENTATION=4 mjk's GLUT 2.3 images. 6/?/95

    GLUT_XLIB_IMPLEMENTATION=5 mjk's GLUT 3.0 images. 10/?/95

    GLUT_XLIB_IMPLEMENTATION=7 mjk's GLUT 3.1+ with glutWarpPoiner. 7/24/96

    GLUT_XLIB_IMPLEMENTATION=8 mjk's GLUT 3.1+ with glutWarpPoiner
    and video resize. 1/3/97

    GLUT_XLIB_IMPLEMENTATION=9 mjk's GLUT 3.4 release with early GLUT 4 routines.

    GLUT_XLIB_IMPLEMENTATION=11 Mesa 2.5's GLUT 3.6 release.

    GLUT_XLIB_IMPLEMENTATION=12 mjk's GLUT 3.6 release with early GLUT 4 routines +
    signal handling.

    GLUT_XLIB_IMPLEMENTATION=13 mjk's GLUT 3.7 beta with GameGLUT support.

    GLUT_XLIB_IMPLEMENTATION=14 mjk's GLUT 3.7 beta with f90gl friend interface.

    GLUT_XLIB_IMPLEMENTATION=15 mjk's GLUT 3.7 beta sync'ed with Mesa <GL/glut.h>
**/
#ifndef GLUT_XLIB_IMPLEMENTATION /* Allow this to be overridden. */
#define GLUT_XLIB_IMPLEMENTATION 15
#endif

/* Display mode bit masks. */
#define GLUT_RGB 0
#define GLUT_RGBA GLUT_RGB
#define GLUT_INDEX 1
#define GLUT_SINGLE 0
#define GLUT_DOUBLE 2
#define GLUT_ACCUM 4
#define GLUT_ALPHA 8
#define GLUT_DEPTH 16
#define GLUT_STENCIL 32
#if (GLUT_API_VERSION >= 2)
#define GLUT_MULTISAMPLE 128
#define GLUT_STEREO 256
#endif
#if (GLUT_API_VERSION >= 3)

```

```

#define GLUT_LUMINANCE          512
#endif

/* Mouse buttons. */
#define GLUT_LEFT_BUTTON        0
#define GLUT_MIDDLE_BUTTON      1
#define GLUT_RIGHT_BUTTON       2

/* Mouse button state. */
#define GLUT_DOWN                0
#define GLUT_UP                  1

#if (GLUT_API_VERSION >= 2)
/* function keys */
#define GLUT_KEY_F1              1
#define GLUT_KEY_F2              2
#define GLUT_KEY_F3              3
#define GLUT_KEY_F4              4
#define GLUT_KEY_F5              5
#define GLUT_KEY_F6              6
#define GLUT_KEY_F7              7
#define GLUT_KEY_F8              8
#define GLUT_KEY_F9              9
#define GLUT_KEY_F10             10
#define GLUT_KEY_F11             11
#define GLUT_KEY_F12             12
/* directional keys */
#define GLUT_KEY_LEFT            100
#define GLUT_KEY_UP              101
#define GLUT_KEY_RIGHT           102
#define GLUT_KEY_DOWN            103
#define GLUT_KEY_PAGE_UP         104
#define GLUT_KEY_PAGE_DOWN       105
#define GLUT_KEY_HOME            106
#define GLUT_KEY_END             107
#define GLUT_KEY_INSERT          108
#endif

/* Entry/exit state. */
#define GLUT_LEFT                0
#define GLUT_ENTERED             1

/* Menu usage state. */
#define GLUT_MENU_NOT_IN_USE     0
#define GLUT_MENU_IN_USE        1

/* Visibility state. */
#define GLUT_NOT_VISIBLE         0
#define GLUT_VISIBLE            1

/* Window status state. */
#define GLUT_HIDDEN              0
#define GLUT_FULLY_RETAINED      1
#define GLUT_PARTIALLY_RETAINED  2
#define GLUT_FULLY_COVERED       3

/* Color index component selection values. */
#define GLUT_RED                 0
#define GLUT_GREEN               1
#define GLUT_BLUE                2

#if defined(_WIN32)

```

```

/* Stroke font constants (use these in GLUT program). */
#define GLUT_STROKE_ROMAN      ((void*)0)
#define GLUT_STROKE_MONO_ROMAN ((void*)1)

/* Bitmap font constants (use these in GLUT program). */
#define GLUT_BITMAP_9_BY_15    ((void*)2)
#define GLUT_BITMAP_8_BY_13    ((void*)3)
#define GLUT_BITMAP_TIMES_ROMAN_10 ((void*)4)
#define GLUT_BITMAP_TIMES_ROMAN_24 ((void*)5)
#if (GLUT_API_VERSION >= 3)
#define GLUT_BITMAP_HELVETICA_10 ((void*)6)
#define GLUT_BITMAP_HELVETICA_12 ((void*)7)
#define GLUT_BITMAP_HELVETICA_18 ((void*)8)
#endif
#else
/* Stroke font opaque addresses (use constants instead in source code). */
GLUTAPI void *glutStrokeRoman;
GLUTAPI void *glutStrokeMonoRoman;

/* Stroke font constants (use these in GLUT program). */
#define GLUT_STROKE_ROMAN      (&glutStrokeRoman)
#define GLUT_STROKE_MONO_ROMAN (&glutStrokeMonoRoman)

/* Bitmap font opaque addresses (use constants instead in source code). */
GLUTAPI void *glutBitmap9By15;
GLUTAPI void *glutBitmap8By13;
GLUTAPI void *glutBitmapTimesRoman10;
GLUTAPI void *glutBitmapTimesRoman24;
GLUTAPI void *glutBitmapHelvetica10;
GLUTAPI void *glutBitmapHelvetica12;
GLUTAPI void *glutBitmapHelvetica18;

/* Bitmap font constants (use these in GLUT program). */
#define GLUT_BITMAP_9_BY_15    (&glutBitmap9By15)
#define GLUT_BITMAP_8_BY_13    (&glutBitmap8By13)
#define GLUT_BITMAP_TIMES_ROMAN_10 (&glutBitmapTimesRoman10)
#define GLUT_BITMAP_TIMES_ROMAN_24 (&glutBitmapTimesRoman24)
#if (GLUT_API_VERSION >= 3)
#define GLUT_BITMAP_HELVETICA_10 (&glutBitmapHelvetica10)
#define GLUT_BITMAP_HELVETICA_12 (&glutBitmapHelvetica12)
#define GLUT_BITMAP_HELVETICA_18 (&glutBitmapHelvetica18)
#endif
#endif

/* glutGet parameters. */
#define GLUT_WINDOW_X          ((GLenum) 100)
#define GLUT_WINDOW_Y          ((GLenum) 101)
#define GLUT_WINDOW_WIDTH      ((GLenum) 102)
#define GLUT_WINDOW_HEIGHT     ((GLenum) 103)
#define GLUT_WINDOW_BUFFER_SIZE ((GLenum) 104)
#define GLUT_WINDOW_STENCIL_SIZE ((GLenum) 105)
#define GLUT_WINDOW_DEPTH_SIZE ((GLenum) 106)
#define GLUT_WINDOW_RED_SIZE    ((GLenum) 107)
#define GLUT_WINDOW_GREEN_SIZE  ((GLenum) 108)
#define GLUT_WINDOW_BLUE_SIZE   ((GLenum) 109)
#define GLUT_WINDOW_ALPHA_SIZE  ((GLenum) 110)
#define GLUT_WINDOW_ACCUM_RED_SIZE ((GLenum) 111)
#define GLUT_WINDOW_ACCUM_GREEN_SIZE ((GLenum) 112)
#define GLUT_WINDOW_ACCUM_BLUE_SIZE ((GLenum) 113)
#define GLUT_WINDOW_ACCUM_ALPHA_SIZE ((GLenum) 114)
#define GLUT_WINDOW_DOUBLEBUFFER ((GLenum) 115)
#define GLUT_WINDOW_RGBA        ((GLenum) 116)

```

```

#define GLUT_WINDOW_PARENT          ((GLenum) 117)
#define GLUT_WINDOW_NUM_CHILDREN    ((GLenum) 118)
#define GLUT_WINDOW_COLORMAP_SIZE   ((GLenum) 119)
#if (GLUT_API_VERSION >= 2)
#define GLUT_WINDOW_NUM_SAMPLES      ((GLenum) 120)
#define GLUT_WINDOW_STEREO           ((GLenum) 121)
#endif
#if (GLUT_API_VERSION >= 3)
#define GLUT_WINDOW_CURSOR           ((GLenum) 122)
#endif
#define GLUT_SCREEN_WIDTH            ((GLenum) 200)
#define GLUT_SCREEN_HEIGHT           ((GLenum) 201)
#define GLUT_SCREEN_WIDTH_MM         ((GLenum) 202)
#define GLUT_SCREEN_HEIGHT_MM        ((GLenum) 203)
#define GLUT_MENU_NUM_ITEMS          ((GLenum) 300)
#define GLUT_DISPLAY_MODE_POSSIBLE   ((GLenum) 400)
#define GLUT_INIT_WINDOW_X           ((GLenum) 500)
#define GLUT_INIT_WINDOW_Y           ((GLenum) 501)
#define GLUT_INIT_WINDOW_WIDTH        ((GLenum) 502)
#define GLUT_INIT_WINDOW_HEIGHT       ((GLenum) 503)
#define GLUT_INIT_DISPLAY_MODE        ((GLenum) 504)
#if (GLUT_API_VERSION >= 2)
#define GLUT_ELAPSED_TIME             ((GLenum) 700)
#endif
#if (GLUT_API_VERSION >= 4 || GLUT_XLIB_IMPLEMENTATION >= 13)
#define GLUT_WINDOW_FORMAT_ID         ((GLenum) 123)
#endif

#if (GLUT_API_VERSION >= 2)
/* glutDeviceGet parameters. */
#define GLUT_HAS_KEYBOARD             ((GLenum) 600)
#define GLUT_HAS_MOUSE                ((GLenum) 601)
#define GLUT_HAS_SPACEBALL            ((GLenum) 602)
#define GLUT_HAS_DIAL_AND_BUTTON_BOX  ((GLenum) 603)
#define GLUT_HAS_TABLET                ((GLenum) 604)
#define GLUT_NUM_MOUSE_BUTTONS         ((GLenum) 605)
#define GLUT_NUM_SPACEBALL_BUTTONS     ((GLenum) 606)
#define GLUT_NUM_BUTTON_BOX_BUTTONS    ((GLenum) 607)
#define GLUT_NUM_DIALS                 ((GLenum) 608)
#define GLUT_NUM_TABLET_BUTTONS        ((GLenum) 609)
#endif
#if (GLUT_API_VERSION >= 4 || GLUT_XLIB_IMPLEMENTATION >= 13)
#define GLUT_DEVICE_IGNORE_KEY_REPEAT  ((GLenum) 610)
#define GLUT_DEVICE_KEY_REPEAT         ((GLenum) 611)
#define GLUT_HAS_JOYSTICK              ((GLenum) 612)
#define GLUT_OWNS_JOYSTICK             ((GLenum) 613)
#define GLUT_JOYSTICK_BUTTONS          ((GLenum) 614)
#define GLUT_JOYSTICK_AXES             ((GLenum) 615)
#define GLUT_JOYSTICK_POLL_RATE        ((GLenum) 616)
#endif

#if (GLUT_API_VERSION >= 3)
/* glutLayerGet parameters. */
#define GLUT_OVERLAY_POSSIBLE          ((GLenum) 800)
#define GLUT_LAYER_IN_USE              ((GLenum) 801)
#define GLUT_HAS_OVERLAY               ((GLenum) 802)
#define GLUT_TRANSPARENT_INDEX         ((GLenum) 803)
#define GLUT_NORMAL_DAMAGED            ((GLenum) 804)
#define GLUT_OVERLAY_DAMAGED           ((GLenum) 805)

#if (GLUT_API_VERSION >= 4 || GLUT_XLIB_IMPLEMENTATION >= 9)
/* glutVideoResizeGet parameters. */

```

```

#define GLUT_VIDEO_RESIZE_POSSIBLE ((GLenum) 900)
#define GLUT_VIDEO_RESIZE_IN_USE   ((GLenum) 901)
#define GLUT_VIDEO_RESIZE_X_DELTA  ((GLenum) 902)
#define GLUT_VIDEO_RESIZE_Y_DELTA  ((GLenum) 903)
#define GLUT_VIDEO_RESIZE_WIDTH_DELTA ((GLenum) 904)
#define GLUT_VIDEO_RESIZE_HEIGHT_DELTA ((GLenum) 905)
#define GLUT_VIDEO_RESIZE_X        ((GLenum) 906)
#define GLUT_VIDEO_RESIZE_Y        ((GLenum) 907)
#define GLUT_VIDEO_RESIZE_WIDTH    ((GLenum) 908)
#define GLUT_VIDEO_RESIZE_HEIGHT   ((GLenum) 909)
#endif

/* glutUseLayer parameters. */
#define GLUT_NORMAL ((GLenum) 0)
#define GLUT_OVERLAY ((GLenum) 1)

/* glutGetModifiers return mask. */
#define GLUT_ACTIVE_SHIFT 1
#define GLUT_ACTIVE_CTRL 2
#define GLUT_ACTIVE_ALT 4

/* glutSetCursor parameters. */
/* Basic arrows. */
#define GLUT_CURSOR_RIGHT_ARROW 0
#define GLUT_CURSOR_LEFT_ARROW 1
/* Symbolic cursor shapes. */
#define GLUT_CURSOR_INFO 2
#define GLUT_CURSOR_DESTROY 3
#define GLUT_CURSOR_HELP 4
#define GLUT_CURSOR_CYCLE 5
#define GLUT_CURSOR_SPRAY 6
#define GLUT_CURSOR_WAIT 7
#define GLUT_CURSOR_TEXT 8
#define GLUT_CURSOR_CROSSHAIR 9
/* Directional cursors. */
#define GLUT_CURSOR_UP_DOWN 10
#define GLUT_CURSOR_LEFT_RIGHT 11
/* Sizing cursors. */
#define GLUT_CURSOR_TOP_SIDE 12
#define GLUT_CURSOR_BOTTOM_SIDE 13
#define GLUT_CURSOR_LEFT_SIDE 14
#define GLUT_CURSOR_RIGHT_SIDE 15
#define GLUT_CURSOR_TOP_LEFT_CORNER 16
#define GLUT_CURSOR_TOP_RIGHT_CORNER 17
#define GLUT_CURSOR_BOTTOM_RIGHT_CORNER 18
#define GLUT_CURSOR_BOTTOM_LEFT_CORNER 19
/* Inherit from parent window. */
#define GLUT_CURSOR_INHERIT 100
/* Blank cursor. */
#define GLUT_CURSOR_NONE 101
/* Fullscreen crosshair (if available). */
#define GLUT_CURSOR_FULL_CROSSHAIR 102
#endif

/* GLUT initialization sub-API. */
GLUTAPI void APIENTRY glutInit(int *argcp, char **argv);
#if defined(_WIN32) && !defined(GLUT_DISABLE_ATEXIT_HACK)
GLUTAPI void APIENTRY __glutInitWithExit(int *argcp, char **argv, void (__cdecl
*exitfunc)(int));
#endif
#ifdef GLUT_BUILDING_LIB
static void APIENTRY glutInit_ATEXIT_HACK(int *argcp, char **argv) {
__glutInitWithExit(argcp, argv, exit); }

```

```

#define glutInit glutInit_ATEXIT_HACK
#endif
#endif
GLUTAPI void APIENTRY glutInitDisplayMode(unsigned int mode);
#if (GLUT_API_VERSION >= 4 || GLUT_XLIB_IMPLEMENTATION >= 9)
GLUTAPI void APIENTRY glutInitDisplayString(const char *string);
#endif
GLUTAPI void APIENTRY glutInitWindowPosition(int x, int y);
GLUTAPI void APIENTRY glutInitWindowSize(int width, int height);
GLUTAPI void APIENTRY glutMainLoop(void);

/* GLUT window sub-API. */
GLUTAPI int APIENTRY glutCreateWindow(const char *title);
#if defined(_WIN32) && !defined(GLUT_DISABLE_ATEXIT_HACK)
GLUTAPI int APIENTRY __glutCreateWindowWithExit(const char *title, void (__cdecl
*exitfunc)(int));
#endif
#ifndef GLUT_BUILDING_LIB
static int APIENTRY glutCreateWindow_ATEXIT_HACK(const char *title) { return
__glutCreateWindowWithExit(title, exit); }
#define glutCreateWindow glutCreateWindow_ATEXIT_HACK
#endif
#endif
GLUTAPI int APIENTRY glutCreateSubWindow(int win, int x, int y, int width, int
height);
GLUTAPI void APIENTRY glutDestroyWindow(int win);
GLUTAPI void APIENTRY glutPostRedisplay(void);
#if (GLUT_API_VERSION >= 4 || GLUT_XLIB_IMPLEMENTATION >= 11)
GLUTAPI void APIENTRY glutPostWindowRedisplay(int win);
#endif
GLUTAPI void APIENTRY glutSwapBuffers(void);
GLUTAPI int APIENTRY glutGetWindow(void);
GLUTAPI void APIENTRY glutSetWindow(int win);
GLUTAPI void APIENTRY glutSetWindowTitle(const char *title);
GLUTAPI void APIENTRY glutSetIconTitle(const char *title);
GLUTAPI void APIENTRY glutPositionWindow(int x, int y);
GLUTAPI void APIENTRY glutReshapeWindow(int width, int height);
GLUTAPI void APIENTRY glutPopWindow(void);
GLUTAPI void APIENTRY glutPushWindow(void);
GLUTAPI void APIENTRY glutIconifyWindow(void);
GLUTAPI void APIENTRY glutShowWindow(void);
GLUTAPI void APIENTRY glutHideWindow(void);
#if (GLUT_API_VERSION >= 3)
GLUTAPI void APIENTRY glutFullScreen(void);
GLUTAPI void APIENTRY glutSetCursor(int cursor);
#endif
#if (GLUT_API_VERSION >= 4 || GLUT_XLIB_IMPLEMENTATION >= 9)
GLUTAPI void APIENTRY glutWarpPointer(int x, int y);
#endif

/* GLUT overlay sub-API. */
GLUTAPI void APIENTRY glutEstablishOverlay(void);
GLUTAPI void APIENTRY glutRemoveOverlay(void);
GLUTAPI void APIENTRY glutUseLayer(GLenum layer);
GLUTAPI void APIENTRY glutPostOverlayRedisplay(void);
#if (GLUT_API_VERSION >= 4 || GLUT_XLIB_IMPLEMENTATION >= 11)
GLUTAPI void APIENTRY glutPostWindowOverlayRedisplay(int win);
#endif
GLUTAPI void APIENTRY glutShowOverlay(void);
GLUTAPI void APIENTRY glutHideOverlay(void);
#endif

/* GLUT menu sub-API. */
GLUTAPI int APIENTRY glutCreateMenu(void (GLUTCALLBACK *func)(int));

```

```

#if defined(_WIN32) && !defined(GLUT_DISABLE_ATEXIT_HACK)
GLUTAPI int APIENTRY __glutCreateMenuWithExit(void (GLUTCALLBACK *func)(int), void
(__cdecl *exitfunc)(int));
#endif
#ifdef GLUT_BUILDING_LIB
static int APIENTRY glutCreateMenu_ATEXIT_HACK(void (GLUTCALLBACK *func)(int)) {
return __glutCreateMenuWithExit(func, exit); }
#define glutCreateMenu glutCreateMenu_ATEXIT_HACK
#endif
#endif

GLUTAPI void APIENTRY glutDestroyMenu(int menu);
GLUTAPI int APIENTRY glutGetMenu(void);
GLUTAPI void APIENTRY glutSetMenu(int menu);
GLUTAPI void APIENTRY glutAddMenuEntry(const char *label, int value);
GLUTAPI void APIENTRY glutAddSubMenu(const char *label, int submenu);
GLUTAPI void APIENTRY glutChangeToMenuEntry(int item, const char *label, int value);
GLUTAPI void APIENTRY glutChangeToSubMenu(int item, const char *label, int submenu);
GLUTAPI void APIENTRY glutRemoveMenuItem(int item);
GLUTAPI void APIENTRY glutAttachMenu(int button);
GLUTAPI void APIENTRY glutDetachMenu(int button);

/* GLUT window callback sub-API. */
GLUTAPI void APIENTRY glutDisplayFunc(void (GLUTCALLBACK *func)(void));
GLUTAPI void APIENTRY glutReshapeFunc(void (GLUTCALLBACK *func)(int width, int
height));
GLUTAPI void APIENTRY glutKeyboardFunc(void (GLUTCALLBACK *func)(unsigned char key,
int x, int y));
GLUTAPI void APIENTRY glutMouseFunc(void (GLUTCALLBACK *func)(int button, int state,
int x, int y));
GLUTAPI void APIENTRY glutMotionFunc(void (GLUTCALLBACK *func)(int x, int y));
GLUTAPI void APIENTRY glutPassiveMotionFunc(void (GLUTCALLBACK *func)(int x, int y));
GLUTAPI void APIENTRY glutEntryFunc(void (GLUTCALLBACK *func)(int state));
GLUTAPI void APIENTRY glutVisibilityFunc(void (GLUTCALLBACK *func)(int state));
GLUTAPI void APIENTRY glutIdleFunc(void (GLUTCALLBACK *func)(void));
GLUTAPI void APIENTRY glutTimerFunc(unsigned int millis, void (GLUTCALLBACK *func)(int
value), int value);
GLUTAPI void APIENTRY glutMenuStateFunc(void (GLUTCALLBACK *func)(int state));
#if (GLUT_API_VERSION >= 2)
GLUTAPI void APIENTRY glutSpecialFunc(void (GLUTCALLBACK *func)(int key, int x, int
y));
GLUTAPI void APIENTRY glutSpaceballMotionFunc(void (GLUTCALLBACK *func)(int x, int y,
int z));
GLUTAPI void APIENTRY glutSpaceballRotateFunc(void (GLUTCALLBACK *func)(int x, int y,
int z));
GLUTAPI void APIENTRY glutSpaceballButtonFunc(void (GLUTCALLBACK *func)(int button,
int state));
GLUTAPI void APIENTRY glutButtonBoxFunc(void (GLUTCALLBACK *func)(int button, int
state));
GLUTAPI void APIENTRY glutDialsFunc(void (GLUTCALLBACK *func)(int dial, int value));
GLUTAPI void APIENTRY glutTabletMotionFunc(void (GLUTCALLBACK *func)(int x, int y));
GLUTAPI void APIENTRY glutTabletButtonFunc(void (GLUTCALLBACK *func)(int button, int
state, int x, int y));
#endif
#if (GLUT_API_VERSION >= 3)
GLUTAPI void APIENTRY glutMenuStatusFunc(void (GLUTCALLBACK *func)(int status, int x,
int y));
GLUTAPI void APIENTRY glutOverlayDisplayFunc(void (GLUTCALLBACK *func)(void));
#endif
#if (GLUT_API_VERSION >= 4 || GLUT_XLIB_IMPLEMENTATION >= 9)
GLUTAPI void APIENTRY glutWindowStatusFunc(void (GLUTCALLBACK *func)(int state));
#endif
#if (GLUT_API_VERSION >= 4 || GLUT_XLIB_IMPLEMENTATION >= 13)
GLUTAPI void APIENTRY glutKeyboardUpFunc(void (GLUTCALLBACK *func)(unsigned char key,
int x, int y));

```

```

GLUTAPI void APIENTRY glutSpecialUpFunc(void (GLUTCALLBACK *func)(int key, int x, int
y));
GLUTAPI void APIENTRY glutJoystickFunc(void (GLUTCALLBACK *func)(unsigned int
buttonMask, int x, int y, int z), int pollInterval);
#endif
#endif
#endif

/* GLUT color index sub-API. */
GLUTAPI void APIENTRY glutSetColor(int, GLfloat red, GLfloat green, GLfloat blue);
GLUTAPI GLfloat APIENTRY glutGetColor(int ndx, int component);
GLUTAPI void APIENTRY glutCopyColormap(int win);

/* GLUT state retrieval sub-API. */
GLUTAPI int APIENTRY glutGet(GLenum type);
GLUTAPI int APIENTRY glutDeviceGet(GLenum type);
#if (GLUT_API_VERSION >= 2)
/* GLUT extension support sub-API */
GLUTAPI int APIENTRY glutExtensionSupported(const char *name);
#endif
#if (GLUT_API_VERSION >= 3)
GLUTAPI int APIENTRY glutGetModifiers(void);
GLUTAPI int APIENTRY glutLayerGet(GLenum type);
#endif

/* GLUT font sub-API */
GLUTAPI void APIENTRY glutBitmapCharacter(void *font, int character);
GLUTAPI int APIENTRY glutBitmapWidth(void *font, int character);
GLUTAPI void APIENTRY glutStrokeCharacter(void *font, int character);
GLUTAPI int APIENTRY glutStrokeWidth(void *font, int character);
#if (GLUT_API_VERSION >= 4 || GLUT_XLIB_IMPLEMENTATION >= 9)
GLUTAPI int APIENTRY glutBitmapLength(void *font, const unsigned char *string);
GLUTAPI int APIENTRY glutStrokeLength(void *font, const unsigned char *string);
#endif

/* GLUT pre-built models sub-API */
GLUTAPI void APIENTRY glutWireSphere(GLdouble radius, GLint slices, GLint stacks);
GLUTAPI void APIENTRY glutSolidSphere(GLdouble radius, GLint slices, GLint stacks);
GLUTAPI void APIENTRY glutWireCone(GLdouble base, GLdouble height, GLint slices, GLint
stacks);
GLUTAPI void APIENTRY glutSolidCone(GLdouble base, GLdouble height, GLint slices,
GLint stacks);
GLUTAPI void APIENTRY glutWireCube(GLdouble size);
GLUTAPI void APIENTRY glutSolidCube(GLdouble size);
GLUTAPI void APIENTRY glutWireTorus(GLdouble innerRadius, GLdouble outerRadius, GLint
sides, GLint rings);
GLUTAPI void APIENTRY glutSolidTorus(GLdouble innerRadius, GLdouble outerRadius, GLint
sides, GLint rings);
GLUTAPI void APIENTRY glutWireDodecahedron(void);
GLUTAPI void APIENTRY glutSolidDodecahedron(void);
GLUTAPI void APIENTRY glutWireTeapot(GLdouble size);
GLUTAPI void APIENTRY glutSolidTeapot(GLdouble size);
GLUTAPI void APIENTRY glutWireOctahedron(void);
GLUTAPI void APIENTRY glutSolidOctahedron(void);
GLUTAPI void APIENTRY glutWireTetrahedron(void);
GLUTAPI void APIENTRY glutSolidTetrahedron(void);
GLUTAPI void APIENTRY glutWireIcosahedron(void);
GLUTAPI void APIENTRY glutSolidIcosahedron(void);

#if (GLUT_API_VERSION >= 4 || GLUT_XLIB_IMPLEMENTATION >= 9)
/* GLUT video resize sub-API. */
GLUTAPI int APIENTRY glutVideoResizeGet(GLenum param);

```

```

GLUTAPI void APIENTRY glutSetupVideoResizing(void);
GLUTAPI void APIENTRY glutStopVideoResizing(void);
GLUTAPI void APIENTRY glutVideoResize(int x, int y, int width, int height);
GLUTAPI void APIENTRY glutVideoPan(int x, int y, int width, int height);

/* GLUT debugging sub-API. */
GLUTAPI void APIENTRY glutReportErrors(void);
#endif

#if (GLUT_API_VERSION >= 4 || GLUT_XLIB_IMPLEMENTATION >= 13)
/* GLUT device control sub-API. */
/* glutSetKeyRepeat modes. */
#define GLUT_KEY_REPEAT_OFF      0
#define GLUT_KEY_REPEAT_ON      1
#define GLUT_KEY_REPEAT_DEFAULT  2

/* Joystick button masks. */
#define GLUT_JOYSTICK_BUTTON_A  1
#define GLUT_JOYSTICK_BUTTON_B  2
#define GLUT_JOYSTICK_BUTTON_C  4
#define GLUT_JOYSTICK_BUTTON_D  8

GLUTAPI void APIENTRY glutIgnoreKeyRepeat(int ignore);
GLUTAPI void APIENTRY glutSetKeyRepeat(int repeatMode);
GLUTAPI void APIENTRY glutForceJoystickFunc(void);

/* GLUT game mode sub-API. */
/* glutGameModeGet. */
#define GLUT_GAME_MODE_ACTIVE           ((GLenum) 0)
#define GLUT_GAME_MODE_POSSIBLE        ((GLenum) 1)
#define GLUT_GAME_MODE_WIDTH           ((GLenum) 2)
#define GLUT_GAME_MODE_HEIGHT           ((GLenum) 3)
#define GLUT_GAME_MODE_PIXEL_DEPTH     ((GLenum) 4)
#define GLUT_GAME_MODE_REFRESH_RATE    ((GLenum) 5)
#define GLUT_GAME_MODE_DISPLAY_CHANGED ((GLenum) 6)

GLUTAPI void APIENTRY glutGameModeString(const char *string);
GLUTAPI int APIENTRY glutEnterGameMode(void);
GLUTAPI void APIENTRY glutLeaveGameMode(void);
GLUTAPI int APIENTRY glutGameModeGet(GLenum mode);
#endif

#ifdef __cplusplus
}
#endif

#ifdef GLUT_APIENTRY_DEFINED
# undef GLUT_APIENTRY_DEFINED
# undef APIENTRY
#endif

#ifdef GLUT_WINGDIAPI_DEFINED
# undef GLUT_WINGDIAPI_DEFINED
# undef WINGDIAPI
#endif

#ifdef GLUT_DEFINED__CDECL
# undef GLUT_DEFINED__CDECL
# undef __cdecl
#endif

```

```
#ifdef GLUT_DEFINED__CRTIMP
# undef GLUT_DEFINED__CRTIMP
# undef _CRTIMP
#endif

#endif          /* __glut_h__ */
```

objectloader.h

```
#ifndef OBJLOAD
#define OBJLOAD
/*****
****
*Loads obj file - limited to vertices, faces, normals, texture maps
*loads to object structure as defined in .h file
****
****/

#define MAX_VERTICES 8000 // Max number of vertices (for each object)
#define MAX_POLYGONS 8000 // Max number of polygons (for each object)
#define MAX_NORMALS 8000 // Max number of polygons (for each object)

// Our vertex type
typedef struct{
    float x,y,z;
}vertex_type;

// Our normal type
typedef struct{
    float i,j,k;
}normcoord_type;

// The polygon (triangle), 3 numbers that aim 3 vertices
typedef struct{
    int v[3],t[3],n[3];
}polygon_type;

// The mapcoord type, 2 texture coordinates for each vertex
typedef struct{
    float u,v;
}mapcoord_type;

// The object type
class object_type{
public:
    int id_texture;
    object_type(){}
    ~object_type(){}
    int objloader(char *p_filename);
    int objdatadisplay();
    void render();
private:
    char name[20];
    int vertices_qty;
    int polygons_qty;
    int mapcoord_qty;
    int normcoord_qty;

    vertex_type vertex[MAX_VERTICES];
    mapcoord_type mapcoord[MAX_VERTICES];
    normcoord_type normcoord[MAX_NORMALS];
    polygon_type polygon[MAX_POLYGONS];
};

#endif
```

/*****

Main.cpp

```
/* Defines
*****
*****/
#define BITMAP_ID 0x4D42          // the universal bitmap ID
#define M_PI      3.14159265358979323846 /* Value of pi */
#define T_WALL 1
#define T_FLOOR 0
#define T_ART1 2
#define T_ART2 3
#define T_ART3 4
#define T_ART4 5
#define T_ART5 6
#define T_ART6 7
#define NUMBEROFTEXTURES 100

/* Includes
*****
*****/
#include <math.h>
#include <iostream>
#include <stdlib.h>
#include "glut.h"
#include "objloader.h"
#include <windows.h>

using namespace std;

/* Texture Information
*****
*****/
BITMAPINFOHEADER bitmapInfoHeader; // bitmap info header (needs windows.h to
define type
unsigned char*      bitmapData;      // the texture data

int screen_width = 640;
int screen_height = 480;

static GLuint      texnum[NUMBEROFTEXTURES]; // array of 2 texture
object names
static GLfloat spin = 0.0;
static GLfloat spin_speed = 1.0;

object_type *objarray[20]; //objects container for our world. Used throughout so
global

//Lights settings
GLfloat light_ambient[] = { 0.1f, 0.1f, 0.1f, 0.1f };
GLfloat light_diffuse[] = { 1.0f, 1.0f, 1.0f, 0.0f };
GLfloat light_specular[] = { 1.0f, 1.0f, 1.0f, 0.0f };
GLfloat light_position[] = { 100.0f, 0.0f, -10.0f, 1.0f };

//Materials settings
GLfloat mat_ambient[] = { 0.5f, 0.5f, 0.0f, 0.0f };
GLfloat mat_diffuse[] = { 0.5f, 0.5f, 0.0f, 0.0f };
GLfloat mat_specular[] = { 1.0f, 1.0f, 1.0f, 0.0f };
GLfloat mat shininess[] = { 1.0f };

GLfloat ambientLight_Tex[] = { 1.0f, 1.0f, 1.0f, 1.0f }; // ambient light
GLfloat diffuseLight_Tex[] = { 1.0f, 1.0f, 1.0f, 1.0f }; // diffuse light
GLfloat specularLight_Tex[] = { 5.0f, 10.0f, .50f, 1.0f }; // diffuse light
GLfloat lightPosition_Tex[] = { 10.0f, 100.0f, 10.0f, 0.0f }; // the light position
```

```

GLfloat light_ambient_obj[] = { 0.1f, 0.1f, 0.1f, 0.1f };
GLfloat light_diffuse_obj[] = { 1.0f, 1.0f, 1.0f, 0.0f };
GLfloat light_specular_obj[] = { 1.0f, 1.0f, 1.0f, 0.0f };
GLfloat light_position_obj[] = { 100.0f, 0.0f, -10.0f, 1.0f };

GLfloat mat_ambient_obj[] = { 0.5f, 0.5f, 0.0f, 0.0f };
GLfloat mat_diffuse_obj[] = { 0.5f, 0.5f, 0.0f, 0.0f };
GLfloat mat_specular_obj[] = { 1.0f, 1.0f, 1.0f, 0.0f };
GLfloat mat_shininess_obj[] = { 1.0f };

/*Bitmap information (from the window.h to define the type of texture data)*/

/*Global Variables*/
float globangle = 0;

/* function prototypes
*****
*****/
unsigned char *LoadBitmapFile(char *filename, BITMAPINFOHEADER *bitmapInfoHeader);
void init(void);
void initTextures(void);
void initPaintings(void);
void initObjects(void);
void display(void);
void spinDisplay(void);
void drawBox(float w, float h, float l, int textureIndex, bool door);
void reshape(int w, int h);
void camera(float, float, float);
void cameraView(float, float, float, float, float, float, float, float);
double toDegrees(double);
void drawRoom(float PointOfOriginX, float PointOfOriginY, float PointOfOriginZ, float
wallLength, float wallThickness, bool insideWall);
void drawPainting1(int textureIndex);
void drawPainting2(int textureIndex);
void drawPainting3(int textureIndex);
void resize(int p_width, int p_height);
void keyboard(unsigned char key, int x, int y);
/* Material variables with the Red Green Blue Values */

//Material
variables
float matAmbient[] = { 0.3f, 0.5f, 0.8f, 1.0f };
float matDiff[] = { 0.25f, 0.25f, 0.25f, 1.0f };
float matAmbient1[] = { 0.83f, 0.5f, 0.8f, 1.0f };
float matDiff1[] = { 0.525f, 0.25f, 0.25f, 1.0f };
float matAmbient2[] = { 0.83f, 0.0f, 0.0f, 1.0f };
float matDiff2[] = { 0.525f, 0.2f, 0.2f, 1.0f };

/* Material variables with the Red Green Blue Values for a white value*/
float matAmbientw[] = { 1.0f, 1.0f, 1.0f, 1.0f };
float matDiffw[] = { 0.25f, 0.25f, 0.25f, 1.0f };

/*****
*
* SUBROUTINE init(void)
*
* Used to initialize OpenGL and to setup our world
*
*****/
void init(void)

```

```

{
    glClearColor(1.0f, 1.0f, 1.0f, 1.0f);

    // Viewport
    transformation
    glViewport(0, 0, screen_width, screen_height);

    // Projection transformation
    glMatrixMode(GL_PROJECTION); // Specifies which matrix stack is the target for
matrix operations
    glLoadIdentity(); // We initialize the projection matrix as identity
    gluPerspective(45.0f, (GLfloat)screen_width / (GLfloat)screen_height, 5.0f,
10000.0f); // We define the "viewing volume"

    //Lights
    initialization and activation
    glLightfv(GL_LIGHT1, GL_AMBIENT, ambientLight_Tex);
    glLightfv(GL_LIGHT1, GL_DIFFUSE, diffuseLight_Tex);
    glLightfv(GL_LIGHT1, GL_DIFFUSE, specularLight_Tex);
    glLightfv(GL_LIGHT1, GL_POSITION, lightPosition_Tex);

    glLightfv(GL_LIGHT2, GL_AMBIENT, light_ambient_obj);
    glLightfv(GL_LIGHT2, GL_DIFFUSE, light_diffuse_obj);
    glLightfv(GL_LIGHT2, GL_DIFFUSE, light_specular_obj);
    glLightfv(GL_LIGHT2, GL_POSITION, light_position_obj);

    glEnable(GL_LIGHT2);
    glEnable(GL_LIGHT1);
    glEnable(GL_LIGHTING);

    //Materials initialization and activation
    glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
    glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
    glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_specular);
    glMaterialfv(GL_FRONT, GL_POSITION, mat_shininess);

    //Other initializations
    glShadeModel(GL_SMOOTH); // Type of shading for the polygons
    glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST); // Texture mapping
perspective correction
    glEnable(GL_TEXTURE_2D); // Texture mapping ON
    glPolygonMode(GL_FRONT_AND_BACK, GL_FILL); // Polygon rasterization mode
(polygons filled)
    glEnable(GL_CULL_FACE); // Enable the back face culling
    glEnable(GL_DEPTH_TEST); // Enable the depth test

    glMaterialfv(GL_FRONT, GL_AMBIENT, matAmbientw);
    glMaterialfv(GL_FRONT, GL_DIFFUSE, matDiffw);

    // Now setup LIGHT0
    glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient_obj); // setup the
ambient element
    glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse_obj); // the diffuse
element
    glLightfv(GL_LIGHT1, GL_SPECULAR, light_specular_obj);
    glLightfv(GL_LIGHT0, GL_POSITION, light_position_obj); // place the light in
the world // Enable
the light

```

```

        //glEnable(GL_LIGHT0);
        //glEnable(GL_LIGHT1);
        /*Lighting*/
        //glEnable(GL_SMOOTH);
        //glEnable(GL_LIGHTING);
        //glEnable(GL_TEXTURE_2D); // enable 2D texturing
        //glPolygonMode(GL_FRONT_AND_BACK, GL_FILL); // Polygon rasterization mode
        (polygon filled)

                                                                    /*Other
Initialisation*/
        glShadeModel(GL_SMOOTH); // Type of shading for the polygons
        glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST); // Texture mapping
        perspective correction

// Initialising the objects for the project
/*Apple Object*/
printf("*****\n");
objarray[0] = new (object_type);
objarray[0]->objloader("Apple.obj");
objarray[0]->objdatadisplay();

/*Table Object*/
objarray[1] = new (object_type);
objarray[1]->objloader("Table2.obj");
objarray[1]->objdatadisplay();

/*Fish Object*/
objarray[2] = new (object_type);
objarray[2]->objloader("Fish.obj");
objarray[2]->objdatadisplay();

/*Fan Object*/
objarray[3] = new (object_type);
objarray[3]->objloader("Fan.obj");
objarray[3]->objdatadisplay();

}

/*****
*
* SUBROUTINE resize(int p_width, int p_height)
*
* This routine must be called everytime we resize our window.
*
* Input parameters: p_width = width in pixels of our viewport
*                   p_height = height in pixels of our viewport
*
*****/

void resize(int p_width, int p_height)
{
    if (screen_width == 0 && screen_height == 0) exit(0);
    screen_width = p_width; // We obtain the new screen width values and store it
    screen_height = p_height; // Height value

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT); // We clear both the color
    and the depth buffer so to draw the next frame

```

```

        glViewport(0, 0, screen_width, screen_height); // Viewport transformation

        glMatrixMode(GL_PROJECTION); // Projection transformation
        glLoadIdentity(); // We initialize the projection matrix as identity
        gluPerspective(45.0f, (GLfloat)screen_width / (GLfloat)screen_height, 5.0f,
10000.0f);

        glutPostRedisplay(); // This command redraw the scene (it calls the same
routine of glutDisplayFunc)
    }

/*****
*
* SUBROUTINE display(void)
*
* This is our main rendering subroutine, called each frame
*
*****/

void display(void)
{
    int wallLength = 10;
    int wallThickness = 0.1;
    float pedestal = 3.0;
    float PointOfOriginX = 0.0f, PointOfOriginY = 0.0f, PointOfOriginZ = 0.0f;
    static int rotation_x, rotation_y, rotation_z;
    int rotation_x_increment = 5;
    float scale = 0.75;

    glPushMatrix();
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);           // clear screen
and depth buffer

    drawRoom(PointOfOriginX, PointOfOriginY, PointOfOriginZ, wallLength,
wallThickness, true);
    drawPainting1(T_ART1); /*Bear Painting*/
    drawPainting2(T_ART2); /*Wolf Painting*/
    drawPainting3(T_ART3); /*Lion Painting*/

    /*Fan object */
    glPushMatrix();
    glTranslatef(0.0, 5.5, 0.0);
    glRotatef(rotation_x, 0.0, 1.0, 0.0); // Rotations of the object (the model
matrix is multiplied by the rotation matrices)
    glRotatef(rotation_y, 1.0, 0.0, 0.0);
    glRotatef(rotation_z, 0.0, 0.0, 0.0);
    objarray[3]->render();
    glPopMatrix();

    PointOfOriginX += -10;
    drawRoom(PointOfOriginX, PointOfOriginY, PointOfOriginZ, wallLength,
wallThickness, false);
    rotation_x += 1;

    /*Apple Object*/
    glPushMatrix();
    glTranslatef(0.0, 3.0, -4.0);
    glRotatef(rotation_x, 0.0, 1.0, 0.0); // Rotations of the object (the model
matrix is multiplied by the rotation matrices)
    glRotatef(rotation_y, 1.0, 0.0, 0.0);
    glRotatef(rotation_z, 0.0, 0.0, 0.0);

```

```

        glScalef(scale, scale, scale);
        objarray[0]->render();
        glPopMatrix();

        /*Table Object*/
        glPushMatrix();
        glTranslatef(0.0, 0.0, -4.0);
        objarray[1]->render();
        glPopMatrix();

        scale = 0.25f;
        /*Fish Object */
        glPushMatrix();
        glColor3f(0.0, 1.0, 0.0);
        glTranslatef(-4.0, 1.8, 0.0);
        glRotatef(90, 0.0, 1.0, 0.0);
        glScalef(scale, scale, scale);
        objarray[2]->render();
        glPopMatrix();

        /*Table Object*/
        glPushMatrix();
        glTranslatef(-4.0, 0.0, 0.0);
        glRotatef(90, 0.0, 1.0, 0.0);

        objarray[1]->render();
        glPopMatrix();

        /*Fan object */
        glPushMatrix();
        glTranslatef(0.0, 5.5, 0.0);
        glRotatef(-rotation_x, 0.0, 1.0, 0.0); // Rotations of the object (the model
matrix is multiplied by the rotation matrices)
        glRotatef(-rotation_y, 1.0, 0.0, 0.0);
        glRotatef(rotation_z, 0.0, 0.0, 0.0);
        objarray[3]->render();
        glPopMatrix();

        glPopMatrix();

        glFlush(); //force drawing

        glutSwapBuffers(); // bring backbuffer to foreground
                                //any errors then display error
codes
        GLenum errCode;
        const GLubyte *errString;
        if ((errCode = glGetError()) != GL_NO_ERROR)
        {
            errString = gluErrorString(errCode);
            fprintf(stderr, "OpenGL error : %s\n", errString);
        }
        //glutSwapBuffers(); // In double buffered mode we invert the positions of the
visible buffer and the writing buffer
    }

    /*****
    *
    * The main routine
    *
    *****/

```

```

int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(100, 100);
    glutCreateWindow(argv[0]);
    init();
    initTextures();
    initPaintings();
    initObjects();
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutKeyboardFunc(keyboard);
    glutIdleFunc(spinDisplay);
    glutMainLoop();
    glutMainLoop();

    return(0);
}

/*int main()
{

for (int i=0;i<1;i++)
{
delete objarray[i];

}

return 1;
}
*/

void reshape(int w, int h)
//reshape
//*****
//OpenGL reshape callback is called when the window is resized
{
    glViewport(0, 0, (GLsizei)w, (GLsizei)h); //respecify size of view port to
match new w and h
    glMatrixMode(GL_PROJECTION); //go into projection mode
    glLoadIdentity(); //reset projection matrix
    gluPerspective(60.0, (GLfloat)w / (GLfloat)h, 1.0, 200.0); //redefine
projection matrix to match changes in w and h
    glMatrixMode(GL_MODELVIEW); //switch back to modelview mode
    glLoadIdentity(); //reset modelview matrix
    gluLookAt(0.0, 2.0, 5.0, 0.0, 2.0, -1.0, 0.0, 1.0, 0.0); // note these match the
initial values

    //of the keyboard fn vx,vy,vz, angle
}

void keyboard(unsigned char key, int x, int y)
{
    //This function alters the camera angle and x,y,z position
    // how would you alter the 'w' and 's' key equations to move

```

```

        //the camera in the direction you're looking? Hint -> needs a little bit of
    trig
    // and both a vx and a vz formula will be needed in each 'w' and 's' case.
    static float Px = 0, Py = 2, Pz = 5, Vtorx = 0, Vtorz = -1.0, angle = 0,
    distance = 0.1, upChange = 0.0, upDown = 0.0;
    static float vx = 0, vy = 2, vz = 5;
    switch (key) {

    case 'b' | 'B':
        // shifting camera towards down
        glMatrixMode(GL_MODELVIEW);
        upDown -= 0.1;
        break;
    case 32:
        glMatrixMode(GL_MODELVIEW);
        // Shifting camera up
        upDown += 0.1;
        break;
    case 'Q' | 'q':
        //strafe left
        Px += distance*cos((double)Vtorx - 1.5708); // x +=
        distance*Math.cos(Math.toRadians(eyeAngle+90));
        Pz += -distance*sin((double)Vtorz - 1.5708);
        break;
    case 'E' | 'e':
        //strafe right
        Px += -distance*sin((double)Vtorx + 1.5708); // x +=
        distance*Math.cos(Math.toRadians(eyeAngle+90));
        Pz += distance*cos((double)Vtorz + 1.5708);

        break;
    case 'w' | 'W':
        //move forward along line of sight
        Px = Px + distance*Vtorx;
        Pz = Pz + distance*Vtorz;
        glMatrixMode(GL_MODELVIEW);
        break;
    case 's' | 'S':
        //move backwards along line of sight
        Px = Px - distance*Vtorx;
        Pz = Pz - distance*Vtorz;
        glMatrixMode(GL_MODELVIEW);
        //vz= vz + 0.1;

        break;
    case 'A' | 'a':
        //look leftdd
        angle = angle - 5;
        Vtorx = sin(angle*0.01745); //
        Vtorz = -cos(angle*0.01745); //assume camera default is pointing down
        negative z axis.
        glMatrixMode(GL_MODELVIEW);
        break;
    case 'D' | 'd':
        //look right
        glMatrixMode(GL_MODELVIEW);
        angle = angle + 5;
        Vtorx = sin(angle*0.01745); //
        Vtorz = -cos(angle*0.01745); //assume camera default is pointing down
        negative z axis.
        break;
    case 'z':

```

```

        glMatrixMode(GL_MODELVIEW);
        break;
    case 'k':
        upChange -= 0.1;
        break;

    case 'l':
        upChange += 0.1;
        break;
    case 27:
        exit(0);
        break;
    default:
        break;
}
glLoadIdentity();
cameraView(Px, Py, Pz, Vtorx, Vtorz, angle, upChange, upDown); //use
glulookat() if you want
glutPostRedisplay();
}

void camera(float x, float z, float angle)
{
    glRotatef(angle, 0, 1, 0); //not the inverse transforms and their sequence
    glTranslatef(-x, -2, -z); //what happens if you swap the transform order
                                //visualize it first then try it.
}

void cameraView(float Px, float Py, float Pz, float Vtorx, float Vtorz, float angle,
float upChange, float upDown)
{
    gluLookAt(Px, Py + upDown, Pz, Px + Vtorx, 2.0 + upDown, Pz + Vtorz, upChange,
1.0, 0.0); // note these match the initial values
    cout << "Eye: " << Px << "\t " << 2.0 << "\t " << Pz << endl;
    cout << "Cen: " << Px + Vtorx << " " << 2.0 << " " << Pz + Vtorz << endl;
    cout << "Ups: " << upChange << " " << 1.0 << " " << 0.0 << endl;

    //of the keyboard Px, Pz PVtorx, PVtorz, angle
}

double toDegrees(double radians)
{
    return radians * (180.0 / M_PI);
}

void spinDisplay(void)
{
    spin = spin + spin_speed;
    if (spin>360.0)
    {
        spin = spin - 360.0;
    }
    glutPostRedisplay();
}

void initTextures(void)
{
    // *1st texture*****
    // load our bitmap file

```

```

        bitmapData = LoadBitmapFile("Floor 512.bmp", &bitmapInfoHeader);

        glGenTextures(20, texnum);                // generate 2 texture objects
        glBindTexture(GL_TEXTURE_2D, texnum[T_FLOOR]); // enable our first texture
object

        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);

        // generate the texture image
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, bitmapInfoHeader.biWidth,
                    bitmapInfoHeader.biHeight, 0, GL_RGB, GL_UNSIGNED_BYTE, bitmapData);

        /*2nd texture*****g

        bitmapData = LoadBitmapFile("Bricked 512.bmp", &bitmapInfoHeader);

        glBindTexture(GL_TEXTURE_2D, texnum[T_WALL]); // enable our second
texture object

        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);

        // generate the texture image
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, bitmapInfoHeader.biWidth,
                    bitmapInfoHeader.biHeight, 0, GL_RGB, GL_UNSIGNED_BYTE, bitmapData);
    }

    void initPaintings(void)
    {

        /*Art Texture*/
        /*1st Art work*****/

        bitmapData = LoadBitmapFile("artwork2.bmp", &bitmapInfoHeader);

        glBindTexture(GL_TEXTURE_2D, texnum[T_ART1]); // enable our second
texture object

        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);

        // generate the texture image
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, bitmapInfoHeader.biWidth,
                    bitmapInfoHeader.biHeight, 0, GL_RGB, GL_UNSIGNED_BYTE, bitmapData);

        /*2nd Art work*****/
        bitmapData = LoadBitmapFile("artwork3.bmp", &bitmapInfoHeader);

        glBindTexture(GL_TEXTURE_2D, texnum[T_ART2]); // enable our second
texture object

        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);

        // generate the texture image
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, bitmapInfoHeader.biWidth,
                    bitmapInfoHeader.biHeight, 0, GL_RGB, GL_UNSIGNED_BYTE, bitmapData);

        /*3rd Art work*****/

```

```

        bitmapData = LoadBitmapFile("artwork4.bmp", &bitmapInfoHeader);

        glBindTexture(GL_TEXTURE_2D, texnum[T_ART3]);          // enable our second
texture object

        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);

        // generate the texture image
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, bitmapInfoHeader.biWidth,
            bitmapInfoHeader.biHeight, 0, GL_RGB, GL_UNSIGNED_BYTE, bitmapData);

        /*4th Art work*****/
        bitmapData = LoadBitmapFile("artwork5.bmp", &bitmapInfoHeader);

        glBindTexture(GL_TEXTURE_2D, texnum[T_ART4]);          // enable our second
texture object

        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);

        // generate the texture image
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, bitmapInfoHeader.biWidth,
            bitmapInfoHeader.biHeight, 0, GL_RGB, GL_UNSIGNED_BYTE, bitmapData);

        /*5th Art work*****/
        bitmapData = LoadBitmapFile("artwork6.bmp", &bitmapInfoHeader);

        glBindTexture(GL_TEXTURE_2D, texnum[T_ART5]);          // enable our second
texture object

        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);

        // generate the texture image
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, bitmapInfoHeader.biWidth,
            bitmapInfoHeader.biHeight, 0, GL_RGB, GL_UNSIGNED_BYTE, bitmapData);

        /*6th Art work*****/
        bitmapData = LoadBitmapFile("artwork7.bmp", &bitmapInfoHeader);

        glBindTexture(GL_TEXTURE_2D, texnum[T_ART6]);          // enable our second
texture object

        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);

        // generate the texture image
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, bitmapInfoHeader.biWidth,
            bitmapInfoHeader.biHeight, 0, GL_RGB, GL_UNSIGNED_BYTE, bitmapData);
    }

    void initObjects(void)
    {

        // Viewport transformation
        glViewport(0, 0, screen_width, screen_height);

        // Projection transformation
        glMatrixMode(GL_PROJECTION); // Specifies which matrix stack is the target for
matrix operations
        glLoadIdentity(); // We initialize the projection matrix as identity

```

```

        gluPerspective(45.0f, (GLfloat)screen_width / (GLfloat)screen_height, 5.0f,
10000.0f); // We define the "viewing volume"

        glLightfv(GL_LIGHT2, GL_AMBIENT, light_ambient_obj);
        glLightfv(GL_LIGHT2, GL_DIFFUSE, light_diffuse_obj);
        glLightfv(GL_LIGHT2, GL_SPECULAR, light_specular_obj);
        glLightfv(GL_LIGHT2, GL_POSITION, light_position_obj);
        glEnable(GL_LIGHT2);
        glEnable(GL_LIGHTING);

        glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient_obj);
        glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse_obj);
        glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular_obj);
        glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess_obj);

        /*Loading Objects */
        //for (int i = 0; i<1; i++)
        //{
        //    printf("*****\n");
        //    objarray[i] = new (object_type);
        //    objarray[i]->objloader("Apple.obj");
        //    objarray[i]->objdatadisplay();
        //}

}

void drawBox(float w, float h, float l, int textureIndex, bool door)
//drawBox
//*****
//This function first translates a further x,y,z units then draws a box of
//width w, height h, length l
{
    glPushMatrix();           //save modelview
    w = w / 2.0; h = h / 2.0; l = l / 2.0; //adjust values so centre is in middle
of box

                                /*Draws Top
Face*/

    /******
    */
    glBindTexture(GL_TEXTURE_2D, texnum[textureIndex]); // enable our second
texture object
    glBegin(GL_POLYGON);
    glNormal3f(0.0f, 1.0f, 0.0f);
    glTexCoord2f(0.0f, 0.0f); glVertex3f(w, l, h);
    glTexCoord2f(1.0f, 0.0f); glVertex3f(w, l, -h);
    glTexCoord2f(1.0f, 1.0f); glVertex3f(-w, l, -h);
    glTexCoord2f(0.0f, 1.0f); glVertex3f(-w, l, h);
    glEnd();

    /*Draws Front Face*/
    /******
    */
    glBindTexture(GL_TEXTURE_2D, texnum[textureIndex]);
    glBegin(GL_POLYGON);
    glNormal3f(0.0f, 0.0f, 1.0f);
    glTexCoord2f(0.0f, 0.0f); glVertex3f(w, l, h);
    glTexCoord2f(1.0f, 0.0f); glVertex3f(-w, l, h);
    glTexCoord2f(1.0f, 1.0f); glVertex3f(-w, -l, h);
    glTexCoord2f(0.0f, 1.0f); glVertex3f(w, -l, h);

```

```

        glEnd();

        /*Draws Right Face*/
        /*****
        *****/
        glBindTexture(GL_TEXTURE_2D, texnum[textureIndex]);    // enable our first
texture object
        glBegin(GL_POLYGON);
        glNormal3f(1.0f, 0.0f, 0.0f);
        glTexCoord2f(0.0f, 0.0f); glVertex3f(w, 1, h);
        glTexCoord2f(1.0f, 0.0f); glVertex3f(w, -1, h);
        glTexCoord2f(1.0f, 1.0f); glVertex3f(w, -1, -h);
        glTexCoord2f(0.0f, 1.0f); glVertex3f(w, 1, -h);
        glEnd();

        /*Draws Left Face*/
        /*****
        *****/
        glBindTexture(GL_TEXTURE_2D, texnum[textureIndex]);    // enable our first
texture object
        glBegin(GL_POLYGON);
        glNormal3f(-1.0f, 0.0f, 0.0f);
        glTexCoord2f(0.0f, 0.0f); glVertex3f(-w, 1, h);
        glTexCoord2f(1.0f, 0.0f); glVertex3f(-w, 1, -h);
        glTexCoord2f(1.0f, 1.0f); glVertex3f(-w, -1, -h);
        glTexCoord2f(0.0f, 1.0f); glVertex3f(-w, -1, h);
        glEnd();

        /*Draws Bottom Face*/
        /*****
        *****/
        glBindTexture(GL_TEXTURE_2D, texnum[textureIndex]);
        glBegin(GL_POLYGON);
        glNormal3f(0.0f, -1.0f, 0.0f);
        glTexCoord2f(0.0f, 0.0f); glVertex3f(-w, -1, h);
        glTexCoord2f(1.0f, 0.0f); glVertex3f(-w, -1, -h);
        glTexCoord2f(1.0f, 1.0f); glVertex3f(w, -1, -h);
        glTexCoord2f(0.0f, 1.0f); glVertex3f(w, -1, h);
        glEnd();

        /*Draws Back Face*/
        /*****
        *****/
        glBindTexture(GL_TEXTURE_2D, texnum[textureIndex]);
        glBegin(GL_POLYGON);
        glNormal3f(0.0f, 0.0f, -1.0f);
        glTexCoord2f(0.0f, 0.0f); glVertex3f(w, -1, -h);
        glTexCoord2f(1.0f, 0.0f); glVertex3f(-w, -1, -h);
        glTexCoord2f(1.0f, 1.0f); glVertex3f(-w, 1, -h);
        glTexCoord2f(0.0f, 1.0f); glVertex3f(w, 1, -h);
        glEnd();

        glPopMatrix();    //restore previous modelview matrix so leaving things as you
found them
    }

    void drawRoom(float PointOfOriginX, float PointOfOriginY, float PointOfOriginZ, float
wallLength, float wallThickness, bool insideWall)
    {
        float translateWall = (float)wallLength / 2;
        /*Draws Floor*/

```

```

/*****
*****/
glColor3f(1.0, 0.0, 0.0); //setcolor ignored when lighting turned on!!!!
glTranslatef(PointOfOriginX, PointOfOriginY, PointOfOriginZ); //move by x,y
and z units
glMaterialfv(GL_FRONT, GL_AMBIENT, matAmbientw);
glMaterialfv(GL_FRONT, GL_DIFFUSE, matDiffw);
drawBox(walllength, walllength, wallThickness + 0.5, T_FLOOR, false); //draw
floor

/*Draws A pedestall in the middle of the
room*/

/*****
*/
if (insideWall == true)
{
    /*Draws Left Wall*/

    /*****
    *****/
    glColor3f(1.0, 0.0, 0.0); //set colour ignored when lighting turned
on!!!!
    glPushMatrix(); //save current position/orientation, etc.
    glTranslatef(-translateWall + PointOfOriginX, translateWall +
PointOfOriginY, PointOfOriginX); //move by x,y and z units
    glRotated(180, 0, 0, 1);
    glMaterialfv(GL_FRONT, GL_AMBIENT, matAmbientw); //set backdrop material
properties
    glMaterialfv(GL_FRONT, GL_DIFFUSE, matDiffw);
    drawBox(wallThickness+0.5, 4, 10, T_WALL, false); //draw back drop
    glPopMatrix(); //restore previous "current" position/orientation etc.
}

/*Draws Right Wall*/
/*****
*****/
glColor3f(1.0, 0.0, 0.0); //set colour ignored when lighting turned on!!!!
glPushMatrix(); //save current position/orientation, etc.
glTranslatef(translateWall + PointOfOriginX, translateWall + PointOfOriginY,
PointOfOriginZ); //move by x,y and z units
glMaterialfv(GL_FRONT, GL_AMBIENT, matAmbientw); //set backdrop material
properties
glMaterialfv(GL_FRONT, GL_DIFFUSE, matDiffw);
drawBox(wallThickness, walllength, walllength, T_WALL, false); //draw back
drop
glPopMatrix(); //restore previous "current" position/orientation etc

/*Draws Back Wall*/

/*****
*/
glColor3f(1.0, 0.0, 0.0); //set colour ignored when lighting turned on!!!!
glPushMatrix(); //save current position/orientation, etc.
glTranslatef(0, translateWall, -translateWall); //move by x,y and z units
glMaterialfv(GL_FRONT, GL_AMBIENT, matAmbientw); //set backdrop material
properties
glMaterialfv(GL_FRONT, GL_DIFFUSE, matDiffw);

```

```

        drawBox(wallLength, wallThickness, wallLength, T_WALL, false); //draw back
drop
        glPopMatrix(); //restore previous "current" position/orientation etc

        /*Draws Front wall*/

/*****
*/
        glColor3f(1.0, 0.0, 0.0); //set colour ignored when lighting turned on!!!!
        glPushMatrix(); //save current position/orientation, etc.
        glTranslatef(0, translateWall, translateWall); //move by x,y and z units
        glMaterialfv(GL_FRONT, GL_AMBIENT, matAmbientw); //set backdrop material
properties
        glMaterialfv(GL_FRONT, GL_DIFFUSE, matDiffw);
        drawBox(wallLength, wallThickness, wallLength, T_WALL, true); //draw back drop
        glPopMatrix(); //restore previous "current" position/orientation etc
    }

    /*Bear Painting*/
    void drawPainting1(int textureIndex)
    {
        int wallLength = 10;
        int wallThickness = 0.5;
        float translateWall = (float)wallLength / 2;

        float heightP = 8.8, widthP = 9.5;
        float scale = 0.5;
        float inverseScale = 1 / (float)scale;
        heightP *= (float)scale;
        widthP *= (float)scale;

        glColor3f(1.0, 0.0, 0.0); //set colour ignored when lighting turned on!!!!
        glPushMatrix(); //save current position/orientation, etc.
        glTranslatef(0, 3, -4.99); //move by x,y and z units
        glRotatef(180, 0, 0, 0);
        glMaterialfv(GL_FRONT, GL_AMBIENT, matAmbientw); //set backdrop material
properties
        glMaterialfv(GL_FRONT, GL_DIFFUSE, matDiffw);

        drawBox(widthP, wallThickness, heightP, textureIndex, false); //draw back drop
        glPopMatrix(); //restore previous "current" position/orientation etc
    }
    /*Bear Painting*/

    /*Wolf Painting*/
    void drawPainting2(int textureIndex)
    {
        int wallLength = 10;
        int wallThickness = 0.5;
        float translateWall = (float)wallLength / 2;

        float heightP = 1.67, widthP = 1.25;
        int scale = 3.5;
        float inverseScale = 1 / (float)scale;
        heightP *= (float)scale;
        widthP *= (float)scale;

        glColor3f(1.0, 0.0, 0.0); //set colour ignored when lighting turned on!!!!
        glPushMatrix(); //save current position/orientation, etc.
        glTranslatef(4.99, 3.5, 0.0); //move by x,y and z units
        glRotatef(90, 0, 1, 0);

```

```
        glMaterialfv(GL_FRONT, GL_AMBIENT, matAmbientw); //set backdrop material
properties
        glMaterialfv(GL_FRONT, GL_DIFFUSE, matDiffw);

        drawBox(widthP, wallThickness, heightP, textureIndex, false); //draw back drop
        glPopMatrix(); //restore previous "current" position/orientation etc
    }
```

```

/*Lion Painting*/
void drawPainting3(int textureIndex)
{
    int wallLength = 10;
    int wallThickness = 0.5;
    float translateWall = (float)wallLength / 2;

    float heightP = 8.8, widthP = 9.5;
    int scale = 3.8;
    float inverseScale = 1 / (float)scale;
    heightP *= (float)inverseScale;
    widthP *= (float)inverseScale;

    glColor3f(1.0, 0.0, 0.0); //set colour ignored when lighting turned on!!!!
    glPushMatrix(); //save current position/orientation, etc.
    glTranslatef(0, 3.5, 4.99); //move by x,y and z units
    glRotatef(0, 0, 0, 0);
    glMaterialfv(GL_FRONT, GL_AMBIENT, matAmbientw); //set backdrop material
properties
    glMaterialfv(GL_FRONT, GL_DIFFUSE, matDiffw);

    drawBox(widthP, wallThickness, heightP, textureIndex, false); //draw back drop
    glPopMatrix(); //restore previous "current" position/orientation etc
}

//*****

// LoadBitmapFile
// desc: Returns a pointer to the bitmap image of the bitmap specified
//       by filename. Also returns the bitmap header information.
//       No support for 8-bit bitmaps.
unsigned char *LoadBitmapFile(char *filename, BITMAPINFOHEADER *bitmapInfoHeader)
{
    FILE *filePtr; // the file
pointer
    BITMAPFILEHEADER bitmapFileHeader; // bitmap file header
    unsigned char *bitmapImage; // bitmap image data
    int imageIdx = 0; // image index counter
    unsigned char tempRGB; // swap variable

    //
    open filename in "read binary" mode
    filePtr = fopen(filename, "rb");
    if (filePtr == NULL)
        return NULL;

    // read the bitmap file header
    fread(&bitmapFileHeader, sizeof(BITMAPFILEHEADER), 1, filePtr);

    // verify that this is a bitmap by checking for the universal bitmap id
    if (bitmapFileHeader.bfType != BITMAP_ID)
    {
        fclose(filePtr);
        return NULL;
    }

    // read the bitmap information header
    fread(bitmapInfoHeader, sizeof(BITMAPINFOHEADER), 1, filePtr);

    // move file pointer to beginning of bitmap data
    fseek(filePtr, bitmapFileHeader.bfOffBits, SEEK_SET);
}

```

```

// allocate enough memory for the bitmap image data
bitmapImage = (unsigned char*)malloc(bitmapInfoHeader->biSizeImage);

// verify memory allocation
if (!bitmapImage)
{
    free(bitmapImage);
    fclose(filePtr);
    return NULL;
}

// read in the bitmap image data
fread(bitmapImage, 1, bitmapInfoHeader->biSizeImage, filePtr);

// make sure bitmap image data was read
if (bitmapImage == NULL)
{
    fclose(filePtr);
    return NULL;
}

// swap the R and B values to get RGB since the bitmap color format is in BGR
for (imageIdx = 0; imageIdx < bitmapInfoHeader->biSizeImage; imageIdx += 3)
{
    tempRGB = bitmapImage[imageIdx];
    bitmapImage[imageIdx] = bitmapImage[imageIdx + 2];
    bitmapImage[imageIdx + 2] = tempRGB;
}

// close the file and return the bitmap image data
fclose(filePtr);
return bitmapImage;
}

```

objloader.cpp

```

/*****
****
*Loads obj file - limited to vertices, faces, normals, texture maps
*loads to object structure as defined in .h file
****
****/

#include <stdio.h>
#include <iostream>
#include <io.h>
#include <stdlib.h>
#include "glut.h"

#include "objloader.h"
using namespace std;
void object_type::render()
{
    glBegin(GL_TRIANGLES); // glBegin and glEnd delimit the vertices that define a
    primitive (in our case triangles)
    for (int j = 0; j<polygons_qty; j++)
    {
        if (j == 1400)
        {
        }
        //----- FIRST VERTEX -----
        //Normal coordinates of the first vertex
        glNormal3f(normcoord[polygon[j].n[0] - 1].i,
                  normcoord[polygon[j].n[0] - 1].j,
                  normcoord[polygon[j].n[0] - 1].k);
        // Texture coordinates of the first vertex
        // Coordinates of the first vertex
        glVertex3f(vertex[polygon[j].v[0] - 1].x,
                  vertex[polygon[j].v[0] - 1].y,
                  vertex[polygon[j].v[0] - 1].z);

        //----- SECOND VERTEX -----
        //Normal coordinates of the first vertex
        glNormal3f(normcoord[polygon[j].n[1] - 1].i,
                  normcoord[polygon[j].n[1] - 1].j,
                  normcoord[polygon[j].n[1] - 1].k);
        // Texture coordinates of the first vertex
        // Coordinates of the first vertex
        glVertex3f(vertex[polygon[j].v[1] - 1].x,
                  vertex[polygon[j].v[1] - 1].y,
                  vertex[polygon[j].v[1] - 1].z);

        //----- THIRD VERTEX -----
        //Normal coordinates of the first vertex
        glNormal3f(normcoord[polygon[j].n[2] - 1].i,
                  normcoord[polygon[j].n[2] - 1].j,
                  normcoord[polygon[j].n[2] - 1].k);
        // Texture coordinates of the first vertex

        // Coordinates of the first vertex
        glVertex3f(vertex[polygon[j].v[2] - 1].x,
                  vertex[polygon[j].v[2] - 1].y,
                  vertex[polygon[j].v[2] - 1].z);
    }
    glEnd();
}
```

```

}

/* vertex_type vertex[MAX_VERTICES];
mapcoord_type mapcoord[MAX_VERTICES];
normcoord_type normcoord[MAX_NORMALS];
polygon_type polygon[MAX_POLYGONS];
int id_texture;
};*/

int object_type::objdatadisplay()
{
    int i;
    printf("VERTICES: %d\n", vertices_qty);
    for (i = 0; i < vertices_qty; i++)
    {
        printf("%f %f %f\n", vertex[i].x, vertex[i].y, vertex[i].z);
    }
    printf("NORMALS: %d\n", normcoord_qty);
    for (i = 0; i < normcoord_qty; i++)
    {
        printf("%f %f %f\n", normcoord[i].i, normcoord[i].j, normcoord[i].k);
    }
    printf("MAP COORDS: %d\n", mapcoord_qty);
    for (i = 0; i < mapcoord_qty; i++)
    {
        printf("%f %f\n", mapcoord[i].u, mapcoord[i].v);
    }
    printf("POLYGONS: %d\n", polygons_qty);
    for (i = 0; i < polygons_qty; i++) //for each vertex of polygon (triangle)
    {
        for (int j = 0; j < 3; j++)
        {
            printf("%d:%d/%d/%d\n", i, polygon[i].v[j], polygon[i].t[j],
polygon[i].n[j]);
        }
    }

    return 1;
}

int object_type::objloader(char *p_filename)
{
    int ivertex = 0; //Index variable
    int inormal = 0;
    int ipolygon = 0;
    int imap = 0;
    int counter = 0;
    char string[256];
    FILE *l_file; //File pointer

    char l_char; //Char variable
    // int vertex;
    unsigned short l_face_flags; //Flag that stores some face information

    if ((l_file = fopen(p_filename, "rt")) == NULL) return 0; //Open the file

    while (!feof(l_file)) //Loop to scan the whole file
    {

```

```

fscanf(l_file, "%c", &l_char);
if (l_char == '\n') //read char if '\n' -skip to next and read
    fscanf(l_file, "%c", &l_char);
switch (l_char) //parse
{
default: fgets(string, 256, l_file);
    break;
case 'v': //a vertex or a normal or a text co-ord
    fscanf(l_file, "%c", &l_char);
    switch (l_char)
    {
    case ' ': //a vertex -expect and so read 3 floats next
        fscanf(l_file, "%f %f %f", &vertex[ivertex].x,
&vertex[ivertex].y, &vertex[ivertex].z);

        ivertex++;
        break;
    case 'n': //a normal -expect and so read 3 floats next
        fscanf(l_file, "%f %f %f", &normcoord[inormal].i,
&normcoord[inormal].j, &normcoord[inormal].k);
        inormal++;
        break;
    case 't': //a texture map coord-expect and so read 2 floats next
        fscanf(l_file, "%f %f", &mapcoord[imap].u,
&mapcoord[imap].v);
        imap++;
        break;
    } //end switch
    break;
case 'f': //a face read next assume format is -> f 1/1/1 2/2/2 3/3/3
    for (int i = 0; i<3; i++) //for each vertex of polygon (triangle)
    {
        fscanf(l_file, "%c", &l_char); //read space char - ignore
this
        fscanf(l_file, "%d", &polygon[ipolygon].v[i]); //read
vertex.
        fscanf(l_file, "%c", &l_char); //read space char - ignore
this
        //fscanf(l_file,"%d",&polygon[ipolygon].t[i]); //read text coord.
        fscanf(l_file, "%c", &l_char); //read space char - ignore
this
        fscanf(l_file, "%d", &polygon[ipolygon].n[i]); //read
normal.

    }
    ipolygon++;
    break;
    counter++;
} //end switch
}
fclose(l_file); // Closes the file stream
vertices_qty = ivertex;
polygons_qty = ipolygon;
mapcoord_qty = imap;
normcoord_qty = inormal;

return 1; //if successful
}

```

FINAL GRADE

GENERAL COMMENTS

Instructor

65/100

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8



Comment 1

probably because you need to transform to view space.



Comment 2

correct it then!

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19

PAGE 20

PAGE 21

PAGE 22

PAGE 23

PAGE 24

PAGE 25

PAGE 26

PAGE 27

PAGE 28

PAGE 29

PAGE 30

PAGE 31

PAGE 32

PAGE 33

PAGE 34

PAGE 35

PAGE 36

PAGE 37

PAGE 38

PAGE 39

GRADING FORM: SMU GENERIC_GJ

ROBERT MCINTYRE

65

RESEARCH-INFORMED LITERATURE



Some relevant texts cited. Try to reduce the number of on-line refs you use and increase standard texts. What about OpenGL Programmers Guide? You've clearly investigated the relevant topics and concepts related to this assignment.

6

KNOWLEDGE AND UNDERSTANDING



You've demonstrated a good understanding of the programming concepts and the application of relevant graphics APIs (OGL) as well as the requisite SE and C++ skills. Some issues with your code might have been solved if you investigated transformations in the pipeline further. Similarly with

17

textures and obj loaders.

ANALYSIS



A good interpretation of the assignment brief but it could have been a little more sophisticated in both scope and the addition of extra features.
Documentation of testing included.

13

PRACTICAL APPLICATION AND DEP.



You've produced a working 3D interactive scene with textures, keyboard control and object loading capability. V good.

26

SKILLS FOR PROFESSIONAL PRAC.



Report includes refs and the structure is fine. Refs included.

3