МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ”

Лабораторна робота №3

з дисципліни

Комп’ютерна графіка

Виконав:

студент групи КВ-81

Петренко М. Е.

Київ 2011

*Код програми:*

#include <windows.h> // Header File For Windows

#include <stdio.h> // Header File For Standard Input/Output

#include <gl\gl.h> // Header File For The OpenGL32 Library

#include <gl\glu.h> // Header File For The GLu32 Library

#include <gl\glaux.h> // Header File For The Glaux Library

HDC hDC=NULL; // Private GDI Device Context

HGLRC hRC=NULL; // Permanent Rendering Context

HWND hWnd=NULL; // Holds Our Window Handle

HINSTANCE hInstance; // Holds The Instance Of The Application

bool keys[256]; // Array Used For The Keyboard Routine

bool active=TRUE; // Window Active Flag Set To TRUE By Default

bool fullscreen=TRUE; // Fullscreen Flag Set To Fullscreen Mode By Default

bool light; // Lighting ON/OFF ( NEW )

bool lp; // L Pressed? ( NEW )

bool fp; // F Pressed? ( NEW )

GLfloat xrot; // X Rotation

GLfloat yrot; // Y Rotation

GLfloat xspeed; // X Rotation Speed

GLfloat yspeed; // Y Rotation Speed

GLfloat z=-5.0f; // Depth Into The Screen

GLfloat LightAmbient[]= { 0.5f, 0.5f, 0.5f, 1.0f };

GLfloat LightDiffuse[]= { 1.0f, 1.0f, 1.0f, 1.0f };

GLfloat LightPosition[]= { 0.0f, 0.0f, 2.0f, 1.0f };

GLuint filter; // Which Filter To Use

GLuint texture[3]; // Storage For 3 Textures

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM); // Declaration For WndProc

AUX\_RGBImageRec \*LoadBMP(char \*Filename) // Loads A Bitmap Image

{

FILE \*File=NULL; // File Handle

if (!Filename) // Make Sure A Filename Was Given

{

return NULL; // If Not Return NULL

}

File=fopen(Filename,"r"); // Check To See If The File Exists

if (File) // Does The File Exist?

{

fclose(File); // Close The Handle

return auxDIBImageLoad(Filename); // Load The Bitmap And Return A Pointer

}

return NULL; // If Load Failed Return NULL

}

int LoadGLTextures() // Load Bitmaps And Convert To Textures

{

int Status=FALSE; // Status Indicator

AUX\_RGBImageRec \*TextureImage[1]; // Create Storage Space For The Texture

memset(TextureImage,0,sizeof(void \*)\*1); // Set The Pointer To NULL

// Load The Bitmap, Check For Errors, If Bitmap's Not Found Quit

if (TextureImage[0]=LoadBMP("Data/Crate.bmp"))

{

Status=TRUE; // Set The Status To TRUE

glGenTextures(3, &texture[0]); // Create Three Textures

// Create Nearest Filtered Texture

glBindTexture(GL\_TEXTURE\_2D, texture[0]);

glTexParameteri(GL\_TEXTURE\_2D,GL\_TEXTURE\_MAG\_FILTER,GL\_NEAREST);

glTexParameteri(GL\_TEXTURE\_2D,GL\_TEXTURE\_MIN\_FILTER,GL\_NEAREST);

glTexImage2D(GL\_TEXTURE\_2D, 0, 3, TextureImage[0]->sizeX, TextureImage[0]->sizeY, 0, GL\_RGB, GL\_UNSIGNED\_BYTE, TextureImage[0]->data);

// Create Linear Filtered Texture

glBindTexture(GL\_TEXTURE\_2D, texture[1]);

glTexParameteri(GL\_TEXTURE\_2D,GL\_TEXTURE\_MAG\_FILTER,GL\_LINEAR);

glTexParameteri(GL\_TEXTURE\_2D,GL\_TEXTURE\_MIN\_FILTER,GL\_LINEAR);

glTexImage2D(GL\_TEXTURE\_2D, 0, 3, TextureImage[0]->sizeX, TextureImage[0]->sizeY, 0, GL\_RGB, GL\_UNSIGNED\_BYTE, TextureImage[0]->data);

// Create MipMapped Texture

glBindTexture(GL\_TEXTURE\_2D, texture[2]);

glTexParameteri(GL\_TEXTURE\_2D,GL\_TEXTURE\_MAG\_FILTER,GL\_LINEAR);

glTexParameteri(GL\_TEXTURE\_2D,GL\_TEXTURE\_MIN\_FILTER,GL\_LINEAR\_MIPMAP\_NEAREST);

gluBuild2DMipmaps(GL\_TEXTURE\_2D, 3, TextureImage[0]->sizeX, TextureImage[0]->sizeY, GL\_RGB, GL\_UNSIGNED\_BYTE, TextureImage[0]->data);

}

if (TextureImage[0]) // If Texture Exists

{

if (TextureImage[0]->data) // If Texture Image Exists

{

free(TextureImage[0]->data); // Free The Texture Image Memory

}

free(TextureImage[0]); // Free The Image Structure

}

return Status; // Return The Status

}

GLvoid ReSizeGLScene(GLsizei width, GLsizei height) // Resize And Initialize The GL Window

{

if (height==0) // Prevent A Divide By Zero By

{

height=1; // Making Height Equal One

}

glViewport(0,0,width,height); // Reset The Current Viewport

glMatrixMode(GL\_PROJECTION); // Select The Projection Matrix

glLoadIdentity(); // Reset The Projection Matrix

// Calculate The Aspect Ratio Of The Window

gluPerspective(45.0f,(GLfloat)width/(GLfloat)height,0.1f,100.0f);

glMatrixMode(GL\_MODELVIEW); // Select The Modelview Matrix

glLoadIdentity(); // Reset The Modelview Matrix

}

int InitGL(GLvoid) // All Setup For OpenGL Goes Here

{

if (!LoadGLTextures()) // Jump To Texture Loading Routine

{

return FALSE; // If Texture Didn't Load Return FALSE

}

glEnable(GL\_TEXTURE\_2D); // Enable Texture Mapping

glShadeModel(GL\_SMOOTH); // Enable Smooth Shading

glClearColor(0.0f, 0.0f, 0.0f, 0.5f); // Black Background

glClearDepth(1.0f); // Depth Buffer Setup

glEnable(GL\_DEPTH\_TEST); // Enables Depth Testing

glDepthFunc(GL\_LEQUAL); // The Type Of Depth Testing To Do

glHint(GL\_PERSPECTIVE\_CORRECTION\_HINT, GL\_NICEST); // Really Nice Perspective Calculations

glLightfv(GL\_LIGHT1, GL\_AMBIENT, LightAmbient); // Setup The Ambient Light

glLightfv(GL\_LIGHT1, GL\_DIFFUSE, LightDiffuse); // Setup The Diffuse Light

glLightfv(GL\_LIGHT1, GL\_POSITION,LightPosition); // Position The Light

glEnable(GL\_LIGHT1); // Enable Light One

return TRUE; // Initialization Went OK

}

int DrawGLScene(GLvoid) // Here's Where We Do All The Drawing

{

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT); // Clear The Screen And The Depth Buffer

glLoadIdentity(); // Reset The View

glTranslatef(0.0f,0.0f,z);

glRotatef(xrot,1.0f,0.0f,0.0f);

glRotatef(yrot,0.0f,1.0f,0.0f);

glBindTexture(GL\_TEXTURE\_2D, texture[filter]);

glBegin(GL\_QUADS);

// Front Face

glNormal3f( 0.0f, 0.0f, 1.0f);

glTexCoord2f(0.0f, 0.0f); glVertex3f(-1.0f, -1.0f, 1.0f);

glTexCoord2f(1.0f, 0.0f); glVertex3f( 1.0f, -1.0f, 1.0f);

glTexCoord2f(1.0f, 1.0f); glVertex3f( 1.0f, 1.0f, 1.0f);

glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f, 1.0f, 1.0f);

// Back Face

glNormal3f( 0.0f, 0.0f,-1.0f);

glTexCoord2f(1.0f, 0.0f); glVertex3f(-1.0f, -1.0f, -1.0f);

glTexCoord2f(1.0f, 1.0f); glVertex3f(-1.0f, 1.0f, -1.0f);

glTexCoord2f(0.0f, 1.0f); glVertex3f( 1.0f, 1.0f, -1.0f);

glTexCoord2f(0.0f, 0.0f); glVertex3f( 1.0f, -1.0f, -1.0f);

// Top Face

glNormal3f( 0.0f, 1.0f, 0.0f);

glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f, 1.0f, -1.0f);

glTexCoord2f(0.0f, 0.0f); glVertex3f(-1.0f, 1.0f, 1.0f);

glTexCoord2f(1.0f, 0.0f); glVertex3f( 1.0f, 1.0f, 1.0f);

glTexCoord2f(1.0f, 1.0f); glVertex3f( 1.0f, 1.0f, -1.0f);

// Bottom Face

glNormal3f( 0.0f,-1.0f, 0.0f);

glTexCoord2f(1.0f, 1.0f); glVertex3f(-1.0f, -1.0f, -1.0f);

glTexCoord2f(0.0f, 1.0f); glVertex3f( 1.0f, -1.0f, -1.0f);

glTexCoord2f(0.0f, 0.0f); glVertex3f( 1.0f, -1.0f, 1.0f);

glTexCoord2f(1.0f, 0.0f); glVertex3f(-1.0f, -1.0f, 1.0f);

// Right face

glNormal3f( 1.0f, 0.0f, 0.0f);

glTexCoord2f(1.0f, 0.0f); glVertex3f( 1.0f, -1.0f, -1.0f);

glTexCoord2f(1.0f, 1.0f); glVertex3f( 1.0f, 1.0f, -1.0f);

glTexCoord2f(0.0f, 1.0f); glVertex3f( 1.0f, 1.0f, 1.0f);

glTexCoord2f(0.0f, 0.0f); glVertex3f( 1.0f, -1.0f, 1.0f);

// Left Face

glNormal3f(-1.0f, 0.0f, 0.0f);

glTexCoord2f(0.0f, 0.0f); glVertex3f(-1.0f, -1.0f, -1.0f);

glTexCoord2f(1.0f, 0.0f); glVertex3f(-1.0f, -1.0f, 1.0f);

glTexCoord2f(1.0f, 1.0f); glVertex3f(-1.0f, 1.0f, 1.0f);

glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f, 1.0f, -1.0f);

glEnd();

xrot+=xspeed;

yrot+=yspeed;

return TRUE; // Keep Going

}

GLvoid KillGLWindow(GLvoid) // Properly Kill The Window

{

if (fullscreen) // Are We In Fullscreen Mode?

{

ChangeDisplaySettings(NULL,0); // If So Switch Back To The Desktop

ShowCursor(TRUE); // Show Mouse Pointer

}

if (hRC) // Do We Have A Rendering Context?

{

if (!wglMakeCurrent(NULL,NULL)) // Are We Able To Release The DC And RC Contexts?

{

MessageBox(NULL,"Release Of DC And RC Failed.","SHUTDOWN ERROR",MB\_OK | MB\_ICONINFORMATION);

}

if (!wglDeleteContext(hRC)) // Are We Able To Delete The RC?

{

MessageBox(NULL,"Release Rendering Context Failed.","SHUTDOWN ERROR",MB\_OK | MB\_ICONINFORMATION);

}

hRC=NULL; // Set RC To NULL

}

if (hDC && !ReleaseDC(hWnd,hDC)) // Are We Able To Release The DC

{

MessageBox(NULL,"Release Device Context Failed.","SHUTDOWN ERROR",MB\_OK | MB\_ICONINFORMATION);

hDC=NULL; // Set DC To NULL

}

if (hWnd && !DestroyWindow(hWnd)) // Are We Able To Destroy The Window?

{

MessageBox(NULL,"Could Not Release hWnd.","SHUTDOWN ERROR",MB\_OK | MB\_ICONINFORMATION);

hWnd=NULL; // Set hWnd To NULL

}

if (!UnregisterClass("OpenGL",hInstance)) // Are We Able To Unregister Class

{

MessageBox(NULL,"Could Not Unregister Class.","SHUTDOWN ERROR",MB\_OK | MB\_ICONINFORMATION);

hInstance=NULL; // Set hInstance To NULL

}

}

/\* This Code Creates Our OpenGL Window. Parameters Are: \*

\* title - Title To Appear At The Top Of The Window \*

\* width - Width Of The GL Window Or Fullscreen Mode \*

\* height - Height Of The GL Window Or Fullscreen Mode \*

\* bits - Number Of Bits To Use For Color (8/16/24/32) \*

\* fullscreenflag - Use Fullscreen Mode (TRUE) Or Windowed Mode (FALSE) \*/

BOOL CreateGLWindow(char\* title, int width, int height, int bits, bool fullscreenflag)

{

GLuint PixelFormat; // Holds The Results After Searching For A Match

WNDCLASS wc; // Windows Class Structure

DWORD dwExStyle; // Window Extended Style

DWORD dwStyle; // Window Style

RECT WindowRect; // Grabs Rectangle Upper Left / Lower Right Values

WindowRect.left=(long)0; // Set Left Value To 0

WindowRect.right=(long)width; // Set Right Value To Requested Width

WindowRect.top=(long)0; // Set Top Value To 0

WindowRect.bottom=(long)height; // Set Bottom Value To Requested Height

fullscreen=fullscreenflag; // Set The Global Fullscreen Flag

hInstance = GetModuleHandle(NULL); // Grab An Instance For Our Window

wc.style = CS\_HREDRAW | CS\_VREDRAW | CS\_OWNDC; // Redraw On Size, And Own DC For Window.

wc.lpfnWndProc = (WNDPROC) WndProc; // WndProc Handles Messages

wc.cbClsExtra = 0; // No Extra Window Data

wc.cbWndExtra = 0; // No Extra Window Data

wc.hInstance = hInstance; // Set The Instance

wc.hIcon = LoadIcon(NULL, IDI\_WINLOGO); // Load The Default Icon

wc.hCursor = LoadCursor(NULL, IDC\_ARROW); // Load The Arrow Pointer

wc.hbrBackground = NULL; // No Background Required For GL

wc.lpszMenuName = NULL; // We Don't Want A Menu

wc.lpszClassName = "OpenGL"; // Set The Class Name

if (!RegisterClass(&wc)) // Attempt To Register The Window Class

{

MessageBox(NULL,"Failed To Register The Window Class.","ERROR",MB\_OK|MB\_ICONEXCLAMATION);

return FALSE; // Return FALSE

}

if (fullscreen) // Attempt Fullscreen Mode?

{

DEVMODE dmScreenSettings; // Device Mode

memset(&dmScreenSettings,0,sizeof(dmScreenSettings)); // Makes Sure Memory's Cleared

dmScreenSettings.dmSize=sizeof(dmScreenSettings); // Size Of The Devmode Structure

dmScreenSettings.dmPelsWidth = width; // Selected Screen Width

dmScreenSettings.dmPelsHeight = height; // Selected Screen Height

dmScreenSettings.dmBitsPerPel = bits; // Selected Bits Per Pixel

dmScreenSettings.dmFields=DM\_BITSPERPEL|DM\_PELSWIDTH|DM\_PELSHEIGHT;

// Try To Set Selected Mode And Get Results. NOTE: CDS\_FULLSCREEN Gets Rid Of Start Bar.

if (ChangeDisplaySettings(&dmScreenSettings,CDS\_FULLSCREEN)!=DISP\_CHANGE\_SUCCESSFUL)

{

// If The Mode Fails, Offer Two Options. Quit Or Use Windowed Mode.

if (MessageBox(NULL,"The Requested Fullscreen Mode Is Not Supported By\nYour Video Card. Use Windowed Mode Instead?","NeHe GL",MB\_YESNO|MB\_ICONEXCLAMATION)==IDYES)

{

fullscreen=FALSE; // Windowed Mode Selected. Fullscreen = FALSE

}

else

{

// Pop Up A Message Box Letting User Know The Program Is Closing.

MessageBox(NULL,"Program Will Now Close.","ERROR",MB\_OK|MB\_ICONSTOP);

return FALSE; // Return FALSE

}

}

}

if (fullscreen) // Are We Still In Fullscreen Mode?

{

dwExStyle=WS\_EX\_APPWINDOW; // Window Extended Style

dwStyle=WS\_POPUP; // Windows Style

ShowCursor(FALSE); // Hide Mouse Pointer

}

else

{

dwExStyle=WS\_EX\_APPWINDOW | WS\_EX\_WINDOWEDGE; // Window Extended Style

dwStyle=WS\_OVERLAPPEDWINDOW; // Windows Style

}

AdjustWindowRectEx(&WindowRect, dwStyle, FALSE, dwExStyle); // Adjust Window To True Requested Size

// Create The Window

if (!(hWnd=CreateWindowEx( dwExStyle, // Extended Style For The Window

"OpenGL", // Class Name

title, // Window Title

dwStyle | // Defined Window Style

WS\_CLIPSIBLINGS | // Required Window Style

WS\_CLIPCHILDREN, // Required Window Style

0, 0, // Window Position

WindowRect.right-WindowRect.left, // Calculate Window Width

WindowRect.bottom-WindowRect.top, // Calculate Window Height

NULL, // No Parent Window

NULL, // No Menu

hInstance, // Instance

NULL))) // Dont Pass Anything To WM\_CREATE

{

KillGLWindow(); // Reset The Display

MessageBox(NULL,"Window Creation Error.","ERROR",MB\_OK|MB\_ICONEXCLAMATION);

return FALSE; // Return FALSE

}

static PIXELFORMATDESCRIPTOR pfd= // pfd Tells Windows How We Want Things To Be

{

sizeof(PIXELFORMATDESCRIPTOR), // Size Of This Pixel Format Descriptor

1, // Version Number

PFD\_DRAW\_TO\_WINDOW | // Format Must Support Window

PFD\_SUPPORT\_OPENGL | // Format Must Support OpenGL

PFD\_DOUBLEBUFFER, // Must Support Double Buffering

PFD\_TYPE\_RGBA, // Request An RGBA Format

bits, // Select Our Color Depth

0, 0, 0, 0, 0, 0, // Color Bits Ignored

0, // No Alpha Buffer

0, // Shift Bit Ignored

0, // No Accumulation Buffer

0, 0, 0, 0, // Accumulation Bits Ignored

16, // 16Bit Z-Buffer (Depth Buffer)

0, // No Stencil Buffer

0, // No Auxiliary Buffer

PFD\_MAIN\_PLANE, // Main Drawing Layer

0, // Reserved

0, 0, 0 // Layer Masks Ignored

};

if (!(hDC=GetDC(hWnd))) // Did We Get A Device Context?

{

KillGLWindow(); // Reset The Display

MessageBox(NULL,"Can't Create A GL Device Context.","ERROR",MB\_OK|MB\_ICONEXCLAMATION);

return FALSE; // Return FALSE

}

if (!(PixelFormat=ChoosePixelFormat(hDC,&pfd))) // Did Windows Find A Matching Pixel Format?

{

KillGLWindow(); // Reset The Display

MessageBox(NULL,"Can't Find A Suitable PixelFormat.","ERROR",MB\_OK|MB\_ICONEXCLAMATION);

return FALSE; // Return FALSE

}

if(!SetPixelFormat(hDC,PixelFormat,&pfd)) // Are We Able To Set The Pixel Format?

{

KillGLWindow(); // Reset The Display

MessageBox(NULL,"Can't Set The PixelFormat.","ERROR",MB\_OK|MB\_ICONEXCLAMATION);

return FALSE; // Return FALSE

}

if (!(hRC=wglCreateContext(hDC))) // Are We Able To Get A Rendering Context?

{

KillGLWindow(); // Reset The Display

MessageBox(NULL,"Can't Create A GL Rendering Context.","ERROR",MB\_OK|MB\_ICONEXCLAMATION);

return FALSE; // Return FALSE

}

if(!wglMakeCurrent(hDC,hRC)) // Try To Activate The Rendering Context

{

KillGLWindow(); // Reset The Display

MessageBox(NULL,"Can't Activate The GL Rendering Context.","ERROR",MB\_OK|MB\_ICONEXCLAMATION);

return FALSE; // Return FALSE

}

ShowWindow(hWnd,SW\_SHOW); // Show The Window

SetForegroundWindow(hWnd); // Slightly Higher Priority

SetFocus(hWnd); // Sets Keyboard Focus To The Window

ReSizeGLScene(width, height); // Set Up Our Perspective GL Screen

if (!InitGL()) // Initialize Our Newly Created GL Window

{

KillGLWindow(); // Reset The Display

MessageBox(NULL,"Initialization Failed.","ERROR",MB\_OK|MB\_ICONEXCLAMATION);

return FALSE; // Return FALSE

}

return TRUE; // Success

}

LRESULT CALLBACK WndProc( HWND hWnd, // Handle For This Window

UINT uMsg, // Message For This Window

WPARAM wParam, // Additional Message Information

LPARAM lParam) // Additional Message Information

{

switch (uMsg) // Check For Windows Messages

{

case WM\_ACTIVATE: // Watch For Window Activate Message

{

if (!HIWORD(wParam)) // Check Minimization State

{

active=TRUE; // Program Is Active

}

else

{

active=FALSE; // Program Is No Longer Active

}

return 0; // Return To The Message Loop

}

case WM\_SYSCOMMAND: // Intercept System Commands

{

switch (wParam) // Check System Calls

{

case SC\_SCREENSAVE: // Screensaver Trying To Start?

case SC\_MONITORPOWER: // Monitor Trying To Enter Powersave?

return 0; // Prevent From Happening

}

break; // Exit

}

case WM\_CLOSE: // Did We Receive A Close Message?

{

PostQuitMessage(0); // Send A Quit Message

return 0; // Jump Back

}

case WM\_KEYDOWN: // Is A Key Being Held Down?

{

keys[wParam] = TRUE; // If So, Mark It As TRUE

return 0; // Jump Back

}

case WM\_KEYUP: // Has A Key Been Released?

{

keys[wParam] = FALSE; // If So, Mark It As FALSE

return 0; // Jump Back

}

case WM\_SIZE: // Resize The OpenGL Window

{

ReSizeGLScene(LOWORD(lParam),HIWORD(lParam)); // LoWord=Width, HiWord=Height

return 0; // Jump Back

}

}

// Pass All Unhandled Messages To DefWindowProc

return DefWindowProc(hWnd,uMsg,wParam,lParam);

}

int WINAPI WinMain( HINSTANCE hInstance, // Instance

HINSTANCE hPrevInstance, // Previous Instance

LPSTR lpCmdLine, // Command Line Parameters

int nCmdShow) // Window Show State

{

MSG msg; // Windows Message Structure

BOOL done=FALSE; // Bool Variable To Exit Loop

// Ask The User Which Screen Mode They Prefer

if (MessageBox(NULL,"Would You Like To Run In Fullscreen Mode?", "Start FullScreen?",MB\_YESNO|MB\_ICONQUESTION)==IDNO)

{

fullscreen=FALSE; // Windowed Mode

}

// Create Our OpenGL Window

if (!CreateGLWindow("NeHe's Textures, Lighting & Keyboard Tutorial",640,480,16,fullscreen))

{

return 0; // Quit If Window Was Not Created

}

while(!done) // Loop That Runs While done=FALSE

{

if (PeekMessage(&msg,NULL,0,0,PM\_REMOVE)) // Is There A Message Waiting?

{

if (msg.message==WM\_QUIT) // Have We Received A Quit Message?

{

done=TRUE; // If So done=TRUE

}

else // If Not, Deal With Window Messages

{

TranslateMessage(&msg); // Translate The Message

DispatchMessage(&msg); // Dispatch The Message

}

}

else // If There Are No Messages

{

// Draw The Scene. Watch For ESC Key And Quit Messages From DrawGLScene()

if ((active && !DrawGLScene()) || keys[VK\_ESCAPE]) // Active? Was There A Quit Received?

{

done=TRUE; // ESC or DrawGLScene Signalled A Quit

}

else // Not Time To Quit, Update Screen

{

SwapBuffers(hDC); // Swap Buffers (Double Buffering)

if (keys['L'] && !lp)

{

lp=TRUE;

light=!light;

if (!light)

{

glDisable(GL\_LIGHTING);

}

else

{

glEnable(GL\_LIGHTING);

}

}

if (!keys['L'])

{

lp=FALSE;

}

if (keys['F'] && !fp)

{

fp=TRUE;

filter+=1;

if (filter>2)

{

filter=0;

}

}

if (!keys['F'])

{

fp=FALSE;

}

if (keys[VK\_PRIOR])

{

z-=0.02f;

}

if (keys[VK\_NEXT])

{

z+=0.02f;

}

if (keys[VK\_UP])

{

xspeed-=0.01f;

}

if (keys[VK\_DOWN])

{

xspeed+=0.01f;

}

if (keys[VK\_RIGHT])

{

yspeed+=0.01f;

}

if (keys[VK\_LEFT])

{

yspeed-=0.01f;

}

if (keys[VK\_F1]) // Is F1 Being Pressed?

{

keys[VK\_F1]=FALSE; // If So Make Key FALSE

KillGLWindow(); // Kill Our Current Window

fullscreen=!fullscreen; // Toggle Fullscreen / Windowed Mode

// Recreate Our OpenGL Window

if (!CreateGLWindow("NeHe's Textures, Lighting & Keyboard Tutorial",640,480,16,fullscreen))

{

return 0; // Quit If Window Was Not Created

}

}

}

}

}

// Shutdown

KillGLWindow(); // Kill The Window

return (msg.wParam); // Exit The Program

}