//---+----3----+----2----+----1----+---<>---+----1----+----2----+----3----+----4

#include "testogl.h"

#include "resource.h"

//---+----3----+----2----+----1----+---<>---+----1----+----2----+----3----+----4

float g\_fAng;

HBITMAP g\_hBmp[24]; // handles to global bitmap images...

float XUP[3] = {1,0,0}, XUN[3] = {-1, 0, 0},

YUP[3] = {0,1,0}, YUN[3] = { 0,-1, 0},

ZUP[3] = {0,0,1}, ZUN[3] = { 0, 0,-1},

ORG[3] = {0,0,0};

GLfloat viewangle = 0, tippangle = 0, vertangle = 0, traj[120][3];

GLfloat d[3] = {0.1, 0.1, 0.1};

GLfloat fL[] = {0.0, 0.0, 1.0, 25.0};

GLfloat fA[] = {0.0, 0.0, 0.0};

GLfloat fMovL = 1;

GLfloat xAngle = 0.0, yAngle = 0.0, zAngle = 0.0;

GLfloat zoomF = 1.0f;

GLboolean booExit = false;

// GLboolean bTest = false;

// GLboolean bLight = false;

//---+----3----+----2----+----1----+---<>---+----1----+----2----+----3----+----4

void SetFAng(float fAng)

{

g\_fAng = fAng;

}

//---+----3----+----2----+----1----+---<>---+----1----+----2----+----3----+----4

void Special\_Keys(int key, int x, int y)

{

switch (key) {

/\*

case GLUT\_KEY\_LEFT : if(glutGetModifiers()==GLUT\_ACTIVE\_CTRL) fA[0]-=fMovL; else viewangle -= 5; break;

case GLUT\_KEY\_RIGHT: if(glutGetModifiers()==GLUT\_ACTIVE\_CTRL) fA[0]+=fMovL; else viewangle += 5; break;

case GLUT\_KEY\_UP : if(glutGetModifiers()==GLUT\_ACTIVE\_CTRL) fA[1]-=fMovL; else tippangle -= 5; break;

case GLUT\_KEY\_DOWN : if(glutGetModifiers()==GLUT\_ACTIVE\_CTRL) fA[1]+=fMovL; else tippangle += 5; break;

case GLUT\_KEY\_PAGE\_UP: if(glutGetModifiers()==GLUT\_ACTIVE\_CTRL) fA[2]-=fMovL; else vertangle += 5; break;

case GLUT\_KEY\_PAGE\_DOWN: if(glutGetModifiers()==GLUT\_ACTIVE\_CTRL) fA[2]+=fMovL; else vertangle -= 5; break;

case GLUT\_KEY\_F2: booTest = true; break;

default: printf (" Special key %c == %d\n", key, key);

\*/

}

// glutPostRedisplay();

}

//---+----3----+----2----+----1----+---<>---+----1----+----2----+----3----+----4

void Keyboard (unsigned char key, int x, int y)

{

switch (key) {

case 'j' : d[0] += 0.1; break;

case 'k' : d[1] += 0.1; break;

case 'l' : d[2] += 0.1; break;

case 'u' : d[0] -= 0.1; break;

case 'i' : d[1] -= 0.1; break;

case 'o' : d[2] -= 0.1; break;

case 'q' : xAngle += 5; break;

case 'w' : yAngle += 5; break;

case 'e' : zAngle += 5; break;

case 'a' : xAngle -= 5; break;

case 's' : yAngle -= 5; break;

case 'd' : zAngle -= 5; break;

case 'f': zoomF+=0.025f; break;

case 'r': zoomF-=0.025f; break;

case 27 : exit(0);

default: printf (" Keyboard %c == %d\n", key, key);

}

// glutPostRedisplay();

}

//---+----3----+----2----+----1----+---<>---+----1----+----2----+----3----+----4

void Triad (void)

{

glColor3f (0.5, 0.5, 0.5);

glBegin (GL\_LINES);

glVertex3fv (ORG); glVertex3fv (XUP);

glVertex3fv (ORG); glVertex3fv (YUP);

glVertex3fv (ORG); glVertex3fv (ZUP);

glEnd ();

glRasterPos3f (1.1, 0.0, 0.0);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

glutBitmapCharacter (GLUT\_BITMAP\_HELVETICA\_18, 'X');

glRasterPos3f (0.0, 1.1, 0.0);

glutBitmapCharacter (GLUT\_BITMAP\_HELVETICA\_18, 'Y');

glRasterPos3f (0.0, 0.0, 1.1);

glutBitmapCharacter (GLUT\_BITMAP\_HELVETICA\_18, 'Z');

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

}

//---+----3----+----2----+----1----+---<>---+----1----+----2----+----3----+----4

void Draw\_Box (void)

{

glBegin (GL\_QUADS);

glColor3f ( 0.0, 0.7, 0.1); // Front - green // original

glVertex3f (-1.0, 1.0, 1.0);

glColor3f ( 0.3, 0.2, 0.1);

glVertex3f ( 1.0, 1.0, 1.0);

glColor3f ( 0.8, 0.0, 0.0);

glVertex3f ( 1.0, -1.0, 1.0);

glColor3f ( 0.0, 0.9, 0.0);

glVertex3f (-1.0, -1.0, 1.0);

glColor3f ( 0.9, 1.0, 0.0); // Back - yellow // original

glVertex3f (-1.0, 1.0, -1.0);

glColor3f ( 0.3, 0.2, 0.1);

glVertex3f ( 1.0, 1.0, -1.0);

glColor3f ( 0.3, 0.2, 0.1);

glVertex3f ( 1.0, -1.0, -1.0);

glColor3f ( 0.3, 0.2, 0.1);

glVertex3f (-1.0, -1.0, -1.0);

glColor3f ( 0.2, 0.2, 1.0); // Top - blue // original

glVertex3f (-1.0, 1.0, 1.0);

glColor3f ( 0.3, 0.2, 0.1);

glVertex3f ( 1.0, 1.0, 1.0);

glColor3f ( 0.3, 0.2, 0.1);

glVertex3f ( 1.0, 1.0, -1.0);

glColor3f ( 0.3, 0.2, 0.1);

glVertex3f (-1.0, 1.0, -1.0);

glColor3f ( 0.7, 0.0, 0.1); // Bottom - red

glVertex3f (-1.0, -1.0, 1.0);

glColor3f ( 0.3, 0.2, 0.1);

glVertex3f ( 1.0, -1.0, 1.0);

glColor3f ( 0.3, 0.2, 0.1);

glVertex3f ( 1.0, -1.0, -1.0);

glColor3f ( 0.3, 0.2, 0.1);

glVertex3f (-1.0, -1.0, -1.0);

glEnd();

}

void Draw\_Surface(void)

{

//

GLfloat vertics[] =

{

};

GLfloat fcolors[] =

{

};

GLfloat normals[] =

{

};

GLubyte indices[] =

{

};

glEnableClientState(GL\_VERTEX\_ARRAY);

glEnableClientState(GL\_COLOR\_ARRAY);

glEnableClientState(GL\_NORMAL\_ARRAY);

glVertexPointer(3, GL\_FLOAT, 0, vertics);

glColorPointer(3, GL\_FLOAT, 0, fcolors);

glNormalPointer(GL\_FLOAT, 0, normals);

glDrawElements(GL\_TRIANGLES, 36, GL\_UNSIGNED\_BYTE, indices);

glDisableClientState(GL\_NORMAL\_ARRAY);

glDisableClientState(GL\_COLOR\_ARRAY);

glDisableClientState(GL\_VERTEX\_ARRAY);

}

void DrawCubes(int nRows, int nUnits) // cubes

{

//

GLfloat vertices[] =

{

-1.0, -1.0, -1.0, +1.0, -1.0, -1.0, +1.0, +1.0, -1.0, -1.0, +1.0, -1.0,

-1.0, -1.0, +1.0, +1.0, -1.0, +1.0, +1.0, +1.0, +1.0, -1.0, +1.0, +1.0,

};

GLfloat fcolors[] =

{

0.2, 0.9, 0.5, 0.9, 0.2, 0.9, 0.0, 0.8, 0.4, 0.5, 0.2, 0.0,

0.8, 0.4, 0.1, 0.1, 0.0, 0.4, 0.0, 0.2, 0.3, 0.0, 0.8, 0.5

};

GLfloat fcolors2[] =

{

0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5,

0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5

};

GLfloat fcolors3[] =

{

0.1, 0.5, 0,1, 0.1, 0.5, 0,1, 0.1, 0.5, 0,1, 0.1, 0.5, 0,1,

0.1, 0.5, 0,1, 0.1, 0.5, 0,1, 0.1, 0.5, 0,1, 0.1, 0.5, 0,1

};

GLubyte indices[] =

{

0, 1, 2, 2, 3, 0,

0, 3, 4, 4, 7, 3,

0, 1, 4, 1, 5, 4,

1, 2, 5, 5, 6, 2,

2, 3, 7, 7, 6, 2,

4, 5, 6, 6, 7, 4

};

GLubyte indicet[] =

{

0, 1, 2, 2, 3, 0,

0, 1, 4, 1, 5, 4,

2, 3, 7, 7, 6, 2,

4, 5, 6, 6, 7, 4

};

glEnableClientState(GL\_VERTEX\_ARRAY);

glEnableClientState(GL\_COLOR\_ARRAY);

// drawing blocks:

for(int j=0; j<nRows /\*RANDURI\*/; j++) // how many rows in the scene ...

{

for(int i=0; i<nUnits /\*CUBURI\*/; i++) // howm many elements in a row ...

{

for(int k=0; k<=23; k+=3)

{

vertices[k+0]+=3; // modify just the x;

}

// then draw:

glVertexPointer(3, GL\_FLOAT, 0, vertices);

glColorPointer(3, GL\_FLOAT, 0, fcolors3);

// glDrawElements(GL\_TRIANGLES, 36, GL\_UNSIGNED\_BYTE, indices);

glDrawElements(GL\_TRIANGLES, 36, GL\_UNSIGNED\_BYTE, indices);

}

// then modify the y's:

for(int k=0; k<=23; k+=3)

{

vertices[k+1]+=3; // modify just the y;

}

}

glDisableClientState(GL\_COLOR\_ARRAY);

glDisableClientState(GL\_VERTEX\_ARRAY);

//

}

void StreetLights(int nPairs, float fDist) // lighting spheres imitating street lights

{

float fTest = 3.5092; // MODIFY THIS VALUE BEFORE EVERY DEBUG/ RUN/ COMPILATION

float fLight[] = {+0.0, +0.3, +0.0, +1.0};

float fColor[] = {+1.0, +1.0, +0.4, +1.0};

glEnable(GL\_LIGHTING); // GL\_EMISSION // GL\_LIGHTING

glEnable(GL\_LIGHT0);

glLightfv(GL\_LIGHT0, GL\_DIFFUSE, fColor);

// drawing sphere:

glPushMatrix();

glTranslatef(2.0, -2.0, 4.0);

for(int i = 0; i<=nPairs; i++) // Pairs of L+R street lights

{

glTranslatef(fDist, 0.0, 0.0); // Distance between lights

for(int j=0; j<=1; j++)

{

if(j==0){glTranslatef(0.0, +4.0, 0.0);}else{glTranslatef(0.0, -4.0, 0.0);}

GLUquadric\* pqdrs01;

pqdrs01 = gluNewQuadric(); // a quadric required

if(pqdrs01)

{

gluQuadricNormals(pqdrs01, GLU\_SMOOTH);

// gluQuadricTexture(pqdrs01, GL\_TRUE);

// gluQuadricDrawStyle(pqdrs01, GLU\_SILHOUETTE);

gluSphere(pqdrs01, 0.2f, 10, 10);

}

else

{

// MessageBoxW(NULL, L"no shpere!", L"", MB\_OK);

}

gluDeleteQuadric(pqdrs01);

}

}

glPopMatrix();

glDisable(GL\_LIGHT0);

glDisable(GL\_LIGHTING);

}

void ExampleX()

{ // thanks to Rachel Grey from "cityontherain.com" for this piece of code

//enable textures

glEnable(GL\_TEXTURE\_2D);

LoadBitmapTexture(6);

glTexEnvf(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GL\_REPLACE);

//use GL\_MODULATE instead of GL\_REPLACE if lighting is being used

//draw a square with specified texture coordinates

float xvals[] = {00.0, 00.0, 40.0, 40.0};

float yvals[] = {40.0, 00.0, 00.0, 40.0};

float svals[] = {1, 0, 0, 1};

float tvals[] = {0, 0, 1, 1};

glPolygonMode(GL\_FRONT, GL\_POLYGON);

glBegin(GL\_POLYGON);

for (int i=0; i < 4; i++)

{

glVertex2f(xvals[i], yvals[i]);

glTexCoord2f(svals[i], tvals[i]);

}

glEnd();

glDisable(GL\_TEXTURE\_2D);

}

void AWall()

{

//////////////////////////////////////////////////////////////////

glEnableClientState( GL\_VERTEX\_ARRAY ); // Enable Vertex Arrays

glEnableClientState( GL\_TEXTURE\_COORD\_ARRAY ); // Enable Texture Coord Arrays

float fLight[] = {-3.0, -4.0, +0.0, +1.0};

float vertices[] = {

+1.0f, +0.0f, -1.0f, //0 index

+1.0f, +0.0f, +1.0f, //1

-1.0f, +0.0f, +1.0f, //2

-1.0f, +0.0f, -1.0f, //3

+1.0f, +1.0f, -1.0f, //4

+1.0f, +1.0f, +1.0f, //5

-1.0f, +1.0f, +1.0f, //6

-1.0f, +1.0f, -1.0f //7

};

short indices[] = { 0, 1, 2, 0, 2, 3,

0, 4, 5, 0, 5, 1};

float texture[] = { 0, 0, 0, 2, 2, 2, 2, 0, //... I repeat these 4 coordinate 2 times

0, 0, 0, 2, 2, 2, 2, 0};

glEnable(GL\_TEXTURE\_2D); // ... forgotten!!!

LoadBitmapTexture(7);

glVertexPointer( 3, GL\_FLOAT, 0, vertices ); // Set The Vertex Pointer To Our Vertex Data

glTexCoordPointer( 2, GL\_FLOAT, 0, texture ); // Set The Vertex Pointer To Our TexCoord Data

glDrawElements(GL\_TRIANGLES, 12, GL\_UNSIGNED\_SHORT, indices); // Draw the wall

glDisableClientState( GL\_VERTEX\_ARRAY ); // Enable Vertex Arrays

glDisableClientState( GL\_TEXTURE\_COORD\_ARRAY ); // Enable Texture Coord Arrays

glLightfv(GL\_LIGHT0, GL\_DIFFUSE, fLight); // GL\_DIFFUSE // BUG!! doesnot work

glDisable(GL\_TEXTURE\_2D);

/////////////////////////////////////////////////////////////////////////////////////////////////

}

void TheHall()

{

float fTest = 0.000;

float fVerts[] = {

+00.00, -01.00, +00.00, +00.00, +01.00, +00.00, +04.00, -01.00, +00.00, +04.00, +01.00, +00.00,

+00.00, -01.00, +03.00, +00.00, +01.00, +03.00, +04.00, -01.00, +03.00, +04.00, +01.00, +03.00,

};

float fColor[] = {

+00.20, +00.30, +30.00, +00.20, +00.20, +30.00, +00.20, +00.10, +00.00, +00.20, +00.40, +00.40,

+00.20, +00.90, +30.00, +00.30, +00.40, +30.00, +00.30, +00.00, +00.50, +00.20, +00.10, +00.00,

};

float fCoords[] = {

0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0

};

BYTE fIndex[] = {

0, 1, 2, 2, 3, 1, 0, 2, 4, 4, 6, 2, 4, 5, 6, 6, 7, 5, 1, 5, 3, 3, 7, 5

};

glEnableClientState(GL\_VERTEX\_ARRAY);

glEnableClientState(GL\_COLOR\_ARRAY);

glPushMatrix();

for(int i=0; i<=80; i++)

{

glVertexPointer(3, GL\_FLOAT, 0, fVerts);

if(i==0&&false)

{

/////////////////////////////////////////////

glEnableClientState(GL\_TEXTURE\_COORD\_ARRAY);

glTexCoordPointer(2, GL\_FLOAT, 0, fCoords);

LoadBitmapTexture(6);

glTexEnvf(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GL\_REPLACE);

/////////////////////////////////////////////

}

else

{

glColorPointer(3, GL\_FLOAT, 0, fColor);

}

glDrawElements(GL\_TRIANGLES, 32, GL\_UNSIGNED\_BYTE, fIndex);

// MessageBoxW(NULL, L"here I am!!", L"DEBUG#1004", MB\_OK);

glTranslatef(4.0, 0.0, 0.0);

if(i==0&&false)

{

/////////////////////////////////////////////

glDisableClientState(GL\_TEXTURE\_COORD\_ARRAY);

/////////////////////////////////////////////

}

}

glPopMatrix();

glDisableClientState(GL\_COLOR\_ARRAY);

glDisableClientState(GL\_VERTEX\_ARRAY);

}

void NewHall()

{

float fVerts[] = {

+00.00, -01.00, +00.00, +00.00, +01.00, +00.00, +04.00, -01.00, +00.00, +04.00, +01.00, +00.00,

+00.00, -01.00, +03.00, +00.00, +01.00, +03.00, +04.00, -01.00, +03.00, +04.00, +01.00, +03.00,

};

glEnable(GL\_TEXTURE\_2D);

LoadBitmapTexture(6);

glTexEnvf(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GL\_REPLACE);

//use GL\_MODULATE instead of GL\_REPLACE if lighting is being used

//draw a square with specified texture coordinates

float xvals[] = {0.0, 0.0, 40.0, 40.0};

float yvals[] = {40.0, 0.0, 0.0, 40.0};

float svals[] = {0, 0, 1, 1};

float tvals[] = {1, 0, 0, 1};

glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_POLYGON);

glBegin(GL\_POLYGON);

for (int i=0; i < 4; i++)

{

glVertex2f(xvals[i], yvals[i]);

glTexCoord2f(svals[i], tvals[i]);

}

glEnd();

glDisable(GL\_TEXTURE\_2D);

glPushMatrix();

glPopMatrix();

}

void Scari(int nTrepte) // deseneaza scari

{

float fVerts[] = {

0.0, -1.0, 0.0, 0.0, +1.0, 0.0,

0.2, -1.0, 0.0, 0.2, +1.0, 0.0,

0,2, -1.0, 0.2, 0.2, +1.0, 0.2

};

float fColor[] = {

0.1, 0.8, 0.2, 0.3, 0.5, 0.4,

0.4, 0.7, 0.5, 0.8, 0.3, 0.1,

0.5, 0.4, 0.8, 0.9, 0.0, 0.8

};

BYTE fIndex[] = {

0, 1, 2, 1, 2, 3,

2, 3, 4, 3, 4, 5

};

glEnableClientState(GL\_VERTEX\_ARRAY);

glEnableClientState(GL\_COLOR\_ARRAY);

glPushMatrix();

glTranslatef(81\*4, 0, 0);

for(int i=1; i<=nTrepte; i++)

{

glVertexPointer(3, GL\_FLOAT, 0, fVerts);

glColorPointer(3, GL\_FLOAT, 0, fColor);

glDrawElements(GL\_TRIANGLES, 8, GL\_UNSIGNED\_BYTE, fIndex);

glTranslatef(0.2, 0.0, 0.2);

}

glPopMatrix();

glDisableClientState(GL\_COLOR\_ARRAY);

glDisableClientState(GL\_VERTEX\_ARRAY);

}

void Punte() // deseneaza puntea // draw the deck

{

glPushMatrix();

glTranslatef(81\*4+0.2\*15, 0, 0.2\*15);

ExampleX();

glPopMatrix();

}

void CeilLightsOnHall()

{

float fLight[] = {+0.0, +0.0, -4.0, +1.0};

float fColor[] = {+1.0, +0.9, +0.0, +1.0};

glEnable(GL\_LIGHTING);

glEnable(GL\_LIGHT0);

glLightfv(GL\_LIGHT0, GL\_DIFFUSE, fColor);

glPushMatrix();

glTranslatef(2.0, 0.0, 3.0);

for(int i = 0; i<=80; i++)

{

GLUquadric\* pqdrLamp = gluNewQuadric(); // a quadric required

if(pqdrLamp)

{

gluQuadricNormals(pqdrLamp, GLU\_SMOOTH);

gluSphere(pqdrLamp, 0.2, 8, 8);

}

glTranslatef(4.0, 0.0, 0.0);

}

glPopMatrix();

glDisable(GL\_LIGHT0);

glDisable(GL\_LIGHTING);

}

void RestrictCamOnTheHall(cxCam\* pCam)

{ // temporary solution

if((pCam->x>0.0)&&(pCam->x<81\*4))

{

if(pCam->y<-0.7){pCam->y=-0.68; return;}

if(pCam->y>+0.7){pCam->y=+0.68; return;}

if(pCam->z<+0.3){pCam->z=+0.32; return;}

if(pCam->z>+2.7){pCam->z=+2.68; return;}

}

}

void TestBitmap(HDC hdc)

{ // does not work!!!!!!!!

LPWSTR wszMsg = (LPWSTR)GlobalAlloc(GPTR, 0x400);

HDC hdcB = CreateCompatibleDC(hdc);

HWND hwnd = WindowFromDC(hdc);

// HBITMAP hbm = (HBITMAP)LoadBitmap(GetModuleHandle(NULL), MAKEINTRESOURCE(IDB\_TEX\_101));

// HBITMAP hbm = (HBITMAP)LoadImage(NULL, MAKEINTRESOURCE(IDB\_TEX\_101), IMAGE\_BITMAP, 0, 0, LR\_DEFAULTSIZE);

HBITMAP hbm = (HBITMAP)LoadImage(NULL, "TEX101.BMP", IMAGE\_BITMAP, 128, 128, LR\_LOADFROMFILE);

BITMAP bm;

HBITMAP hbmOld;

PAINTSTRUCT ps;

LPRECT lprcCli = (LPRECT)GlobalAlloc(GPTR, sizeof(RECT));

if(hbm==NULL)

{

MessageBoxW(NULL, wszMsg, L"bitmap not loaded!!!", 0);

}

GetObject(hbm, sizeof(bm), &bm);

GetClientRect(hwnd, lprcCli);

BeginPaint(hwnd, &ps);

hbmOld = (HBITMAP)SelectObject(hdcB, hbm);

BitBlt(hdc, lprcCli->left, lprcCli->top, lprcCli->left+bm.bmWidth, lprcCli->top+bm.bmHeight, hdcB, 0, 0, SRCPAINT);

SelectObject(hdcB, hbmOld);

// DeleteObject(hbmOld);

// DeleteObject(hbm);

EndPaint(hwnd, &ps);

GlobalFree(lprcCli);

CloseHandle(hbmOld);

CloseHandle(hbm);

CloseHandle(hwnd);

GlobalFree(wszMsg);

}

void TestTexture()

{ // texture:

glEnable(GL\_TEXTURE\_2D);

// SetTextureFilter(TF\_NONE);

LoadBitmapTexture(6);

float oneFortieth = 1.0/40.0;

//define how the s parameter depends on x, y, z, w

GLfloat myparamsS[] = {oneFortieth, 0.00, 0.00, 1.0};

//define how the t parameter depends on x, y, z, w

GLfloat myparamsT[] = {0.00, oneFortieth, 0.00, 1.0};

glTexGeni(GL\_S, GL\_TEXTURE\_GEN\_MODE, GL\_OBJECT\_LINEAR);

glTexGenfv(GL\_S, GL\_OBJECT\_PLANE, myparamsS);

glTexGeni(GL\_T, GL\_TEXTURE\_GEN\_MODE, GL\_OBJECT\_LINEAR);

glTexGenfv(GL\_T, GL\_OBJECT\_PLANE, myparamsT);

glEnable(GL\_TEXTURE\_GEN\_S);

glEnable(GL\_TEXTURE\_GEN\_T);

glTexEnvf(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GL\_REPLACE);

//use GL\_MODULATE instead of GL\_REPLACE if lighting is being used

// ~texture

}

//---+----3----+----2----+----1----+---<>---+----1----+----2----+----3----+----4

void Redraw(HDC hdc, BOOL bLight, CxCam\* pCam)

{

// main function that draws all

glClear (GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

glEnable (GL\_DEPTH\_TEST);

glEnable(GL\_POLYGON\_SMOOTH);

glLoadIdentity();

gluLookAt(pCam->x, pCam->y, pCam->z, pCam->x0, pCam->y0, pCam->z0, pCam->xv, pCam->yv, pCam->zv);

// Drawing objects:

// Triad ();

DrawCubes(1, 240); // cubes // CUBURI> Rows, Units

StreetLights(60, 8.0); // lighting spheres // pairs, distance between

// TheHall();

// CeilLightsOnHall();

// RestrictCamOnTheHall(pCam);

Scari(15);

Punte();

// ExampleX();

// AWall();

// TheSky(pCam);

// TheSun(pCam); // shows the sun on the sky// long distance simulated by moving with the camera..

// TheEarth(pCam); // shows the earth, relative to position of the cam, simulating a very long distance...

// ~ End of Drawing objects

SwapBuffers(hdc);

}

void Initializations(float fFovy, float fWidth, float fHeight)

{

glClearColor (0.1, 0.0, 0.1, 0.0); // in space

glMatrixMode (GL\_PROJECTION);

gluPerspective (45.0, 16.0/9.0, 0.1, 10120.0);

glMatrixMode (GL\_MODELVIEW);

if(CxxLogPrnW(L"c:\\tnc\\testogllog.txt", L"Application Initialized")==0)

{

MessageBoxW(NULL, L"Log file error", L"", MB\_OK);

}

for (int i=0; i<=23; i++)

g\_hBmp[i] = NULL;

}

//---+----3----+----2----+----1----+---<>---+----1----+----2----+----3----+----4

void LoadBitmapTexture(int id)

{ // from "http://cityintherain.com/howtotex.html" // thanks to Rachel Grey

HDC hdcY = CreateCompatibleDC(NULL);

HBITMAP hBmp = NULL;

LPWSTR wszTexImg = (LPWSTR)GlobalAlloc(GPTR, 0x60); // for the image...

swprintf(wszTexImg, L"TEX1%02d.BMP", id);

// hBmp = (HBITMAP)LoadImage(GetModuleHandle(NULL), MAKEINTRESOURCE(id), IMAGE\_BITMAP, 0, 0, LR\_CREATEDIBSECTION);

// hBmp = (HBITMAP)LoadImageW(GetModuleHandle(NULL), MAKEINTRESOURCEW(id), IMAGE\_BITMAP, 0, 0, LR\_CREATEDIBSECTION);

hBmp = (HBITMAP)LoadImageW(GetModuleHandle(NULL), wszTexImg, IMAGE\_BITMAP, 0, 0, LR\_LOADFROMFILE);

// hBmp = (HBITMAP)LoadBitmap(GetModuleHandle(NULL), MAKEINTRESOURCE(id));

GlobalFree(wszTexImg);

if(hBmp==INVALID\_HANDLE\_VALUE)

MessageBoxW(NULL, L"Texture bitmap image not loaded", L"", MB\_OK);

//get info about the bitmap

BITMAP bm;

BITMAPINFO bi;

GetObject(hBmp, sizeof(bm), &bm);

LPBYTE lpbytImg = (LPBYTE)GlobalAlloc(GPTR, bm.bmWidth\*bm.bmHeight\*3);

bi.bmiHeader.biSize = sizeof(BITMAPINFOHEADER);

bi.bmiHeader.biWidth = bm.bmWidth;

bi.bmiHeader.biHeight = bm.bmHeight;

bi.bmiHeader.biPlanes = 1;

bi.bmiHeader.biBitCount = 24;

bi.bmiHeader.biCompression = BI\_RGB;

bi.bmiHeader.biSizeImage = 0;

GetDIBits(hdcY, hBmp, 0, bm.bmHeight, lpbytImg, &bi, DIB\_RGB\_COLORS);

//tell OpenGL to ignore padding at ends of rows

// glPixelStorei(GL\_UNPACK\_ALIGNMENT, 4);

glTexParameterf(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_REPEAT);

glTexParameterf(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_REPEAT);

// glTexParameterf(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_NEAREST);

// glTexParameterf(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_NEAREST);

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

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

// glTexEnvf(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GL\_MODULATE);

glTexImage2D(GL\_TEXTURE\_2D, 0, 3, bm.bmWidth, bm.bmHeight, 0, GL\_BGR, GL\_UNSIGNED\_BYTE, lpbytImg);

// LPWSTR wszMsg = (LPWSTR)GlobalAlloc(GPTR, 0x100);

// swprintf(wszMsg, L"%dx%d", BM.bmWidth, BM.bmHeight);

// MessageBoxW(NULL, wszMsg, L"", MB\_OK);

// GlobalFree(wszMsg);

GlobalFree(lpbytImg);

DeleteObject((HGDIOBJ) hBmp); //avoid memory leak (Windows)

}

int MoveIt(cxCam\* pCam, float fds)

{ // moves the camera forward (or backward, if fdx is negative) its direction

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

LPWSTR wszMsg = (LPWSTR)GlobalAlloc(GPTR, 0xff);

swprintf(wszMsg, L"x=%g, x0=%g, y=%g, y0=%g", pCam->x, pCam->x0, pCam->y, pCam->y0);

MessageBoxW(NULL, wszMsg, L"", MB\_OK);

swprintf(wszMsg, L"R=%g, Rp=%g, ffi=%g, fth=%g", R, Rp, ffi, fth);

MessageBoxW(NULL, wszMsg, L"", MB\_OK);

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

float R = (float)sqrt((pCam->x0-pCam->x)\*(pCam->x0-pCam->x)+(pCam->x0-pCam->x)\*(pCam->x0-pCam->x)+(pCam->x0-pCam->x)\*(pCam->x0-pCam->x));

float x = (pCam->x0-pCam->x)/R; // the calculated components of the orientation vector

float y = (pCam->y0-pCam->y)/R;

float z = (pCam->z0-pCam->z)/R;

pCam->x+=fds\*x; // increment with the x-component of "fds"

pCam->y+=fds\*y; // increment with the y-component of "fds"

pCam->z+=fds\*z; // increment with the z-component of "fds"

pCam->x0+=fds\*x; // increment with the x-component of "fds"

pCam->y0+=fds\*y; // increment with the y-component of "fds"

pCam->z0+=fds\*z; // increment with the z-component of "fds"

return 0; // return success

}

int TurnIt(cxCam\* pCam, float fdt, float fdf, float fdh)

{

float R = (float)sqrt((pCam->x0-pCam->x)\*(pCam->x0-pCam->x)+(pCam->y0-pCam->y)\*(pCam->y0-pCam->y)+(pCam->z0-pCam->z)\*(pCam->z0-pCam->z));

// use the components of the normal vector: fdt (delta-theta) rotates about this vector

// guess the components of the orientation vector:

float x = (pCam->x0-pCam->x)/R; // the calculated components of the orientation vector

float y = (pCam->y0-pCam->y)/R;

float z = (pCam->z0-pCam->z)/R;

// guess the components of the normal vector (in fact, they are stored in the pCam structure)

float u = pCam->xv;

float v = pCam->yv;

float w = pCam->zv;

// guess the components of the yawn vector: // in fact, the rotation with 90 degrees of the normal vector about the direction vector

float r = x\*(x\*u+y\*v+z\*w)+u;

float s = y\*(x\*u+y\*v+z\*w)+v;

float t = z\*(x\*u+y\*v+z\*w)+w;

// first pitch the camera (turn it about the normal vector)

float xt = u\*(u\*x+v\*y+w\*z)\*(1-cos(fdt))+x\*cos(fdt)+(-w\*y+v\*z)\*sin(fdt);

float yt = v\*(u\*x+v\*y+w\*z)\*(1-cos(fdt))+y\*cos(fdt)+(+w\*x-u\*z)\*sin(fdt);

float zt = w\*(u\*x+v\*y+w\*z)\*(1-cos(fdt))+z\*cos(fdt)+(-v\*x+u\*y)\*sin(fdt);

x = xt;

y = yt,

z = zt;

pCam->x0 = pCam->x+R\*x;

pCam->y0 = pCam->y+R\*y;

pCam->z0 = pCam->z+R\*z;

// guess the components of the rotated accordingly yaw vector:

r = xt\*(xt\*u+yt\*v+zt\*w)+u;

s = yt\*(xt\*u+yt\*v+zt\*w)+v;

t = zt\*(xt\*u+yt\*v+zt\*w)+w;

// rotate (elevate) now the orientation and the normal vector about the yaw vector:

x = (pCam->x0-pCam->x)/R; // the calculated components of the orientation vector

y = (pCam->y0-pCam->y)/R;

z = (pCam->z0-pCam->z)/R;

xt = r\*(r\*x+s\*y+t\*z)\*(1-cos(fdf))+x\*cos(fdf)+(-w\*y+v\*z)\*sin(fdf);

yt = s\*(r\*x+s\*y+t\*z)\*(1-cos(fdf))+y\*cos(fdf)+(+w\*x-u\*z)\*sin(fdf);

zt = t\*(r\*x+s\*y+t\*z)\*(1-cos(fdf))+z\*cos(fdf)+(-v\*x+u\*y)\*sin(fdf);

x = xt;

y = yt;

z = zt;

pCam->x0 = pCam->x+R\*x;

pCam->y0 = pCam->y+R\*y;

pCam->z0 = pCam->z+R\*z;

return 0; // return success

}

int RotateVector(float u, float v, float w, float\* x, float\* y, float\* z)

{

return 0;

}

int RotateIt(cxCam\* pCam, float fdt, float fdf) // will be replaced!!!

{ // this rotates the camera (only) around the point of view (around the camera position)

// pCam does not need theta and phi that can be deduced from the other.

// Rotation about the origin of the normal veg

pCam->th+=fdt; // increment with delta-theta

pCam->fi+=fdf; // increment with delta-phi

pCam->xv = cos(PI+pCam->th)\*sin(pCam->fi);

pCam->yv = sin(PI+pCam->th)\*sin(pCam->fi);

pCam->zv = cos(pCam->fi);

// pCam->zv = 1+(-2)\*(cos(pCam->fi)<0); // reset the normal vector von upsidedown to downside up, when the camera turns overhead

// if(abs(pCam->fi)>89\*DEG) pCam->fi-=fdf; // limit the vertical elevation to +/- 90 degrees

pCam->x0 = pCam->x+pCam->R\*cos(pCam->th)\*cos(pCam->fi);

pCam->y0 = pCam->y+pCam->R\*sin(pCam->th)\*cos(pCam->fi);

pCam->z0 = pCam->z+pCam->R\*sin(pCam->fi);

return 0;

}

int PrintCoords(cxCam\* pCam, HDC hdc)

{

LPWSTR wszMsg = (LPWSTR)GlobalAlloc(GPTR, 0x400);

float R = (float)sqrt((pCam->x0-pCam->x)\*(pCam->x0-pCam->x)+(pCam->y0-pCam->y)\*(pCam->y0-pCam->y)+(pCam->z0-pCam->z)\*(pCam->z0-pCam->z));

float fi = (float)asin(pCam->z0-pCam->z)/R;

float Rp = (float)sqrt((pCam->x0-pCam->x)\*(pCam->x0-pCam->x)+(pCam->y0-pCam->y)\*(pCam->y0-pCam->y));

float th = (float)acos(pCam->x0-pCam->x)/Rp;

swprintf(wszMsg, L" x: %+08.3f x0: %+08.3f", pCam->x, pCam->x0);

TextOutW(hdc, 4, 28, wszMsg, wcslen(wszMsg));

swprintf(wszMsg, L" y: %+08.3f y0: %+08.3f", pCam->y, pCam->y0);

TextOutW(hdc, 4, 40, wszMsg, wcslen(wszMsg));

swprintf(wszMsg, L" z: %+08.3f z0: %+08.3f", pCam->z, pCam->z0);

TextOutW(hdc, 4, 52, wszMsg, wcslen(wszMsg));

// swprintf(wszMsg, L" cos(phi): %+08.3f", cos(pCam->fi));

// TextOutW(hdc, 4, 64, wszMsg, wcslen(wszMsg));

swprintf(wszMsg, L" th: %+08.3f fi=%+08.3f R : %+08.3f Rp: %+08.3f", th, fi, R, Rp);

TextOutW(hdc, 4, 04, wszMsg, wcslen(wszMsg));

GlobalFree(wszMsg);

return 0;

}

int PrintParams(float fds, float fdq, float fdt, float fdf, HDC hdc)

{// shows the parameters of the command device...

LPWSTR wszMsg = (LPWSTR)GlobalAlloc(GPTR, 0x400);

swprintf(wszMsg, L" fds: %+08.3f fdq: %+08.3f fdt: %+08.3f fdf: %+08.3f", fds, fdq, fdt, fdf);

TextOutW(hdc, 380, 04, wszMsg, wcslen(wszMsg));

return 0;

}

//---+----3----+----2----+----1----+---<>---+----1----+----2----+----3----+----4

void TheSun(cxCam\* pCam) // shows the Sun on the sky

{

float fLight[] = {+1.0, +1.0, +0.0, +0.0};

float fColor[] = {+1.0, +1.0, +0.7, +1.0};

glEnable(GL\_LIGHTING);

glEnable(GL\_LIGHT0);

glLightfv(GL\_LIGHT0, GL\_DIFFUSE, fLight); // GL\_DIFFUSE

glPushMatrix();

glTranslatef(pCam->x-8200.0, pCam->y-1840.8, pCam->z+1880.1);

GLUquadric\* pqdrBall = gluNewQuadric(); // a quadric required

if(pqdrBall)

{

gluQuadricNormals(pqdrBall, GLU\_SMOOTH);

gluSphere(pqdrBall, 420.1, 24, 24);

}

glPopMatrix();

glDisable(GL\_LIGHT0);

glDisable(GL\_LIGHTING);

//

}

void TheEarth(cxCam\* pCam)

{

GLUquadric\* pqABall;

GLUquadric\* pqAtm[3];

glPushMatrix();

glTranslatef(pCam->x+4000.0, pCam->y-900.0, pCam->z-900.0);

pqABall = gluNewQuadric(); // a quadric required

if(pqABall)

{

glEnable(GL\_TEXTURE\_2D);

LoadBitmapTexture(10);

gluQuadricTexture(pqABall, 1);

gluQuadricNormals(pqABall, GLU\_SMOOTH);

gluSphere(pqABall, 2400.0f, 40, 40);

glDisable(GL\_TEXTURE\_2D);

gluDeleteQuadric(pqABall);

}

for(int i=0; i<=0; i++)

{

pqAtm[i] = gluNewQuadric(); // a quadric required

}

if(pqAtm[0]&&FALSE)

{

glColor4f(0.0f, 0.0f, 0.3f, 0.5f);

glBlendFunc(GL\_SRC\_ALPHA, GL\_DST\_ALPHA);

glEnable(GL\_BLEND);

for(int i=0; i<=0; i++)

{

// gluQuadricDrawStyle(pqAtm[i], GLU\_FILL);

gluQuadricNormals(pqAtm[i], GLU\_SMOOTH);

gluSphere(pqAtm[i], 2000.0f+50.0f\*(i+1), 40, 40);

}

glDisable(GL\_BLEND);

gluDeleteQuadric(pqAtm[0]);

}

glPopMatrix();

}

void TheSky(cxCam\* pCam) // draws the sky as a big textured sphere with the center in the point of view

{

GLUquadric\* pqABall = gluNewQuadric();

glPushMatrix();

glTranslatef(pCam->x, pCam->y, pCam->z);

if(pqABall)

{

glEnable(GL\_TEXTURE\_2D);

LoadBitmapTexture(16);

gluQuadricTexture(pqABall, 1);

gluQuadricNormals(pqABall, GLU\_SMOOTH);

gluSphere(pqABall, 2000.0f, 40, 40);

glDisable(GL\_TEXTURE\_2D);

gluDeleteQuadric(pqABall);

}

glPopMatrix();

}

void TheSea() // draws seawater / textured

{

}

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////