~\Desktop\gravity.cpp

```
1  //-----------------------------------------------------------------------------
2  // File: gravity.cpp
3  //
4  // Copyright (c) 1999 Microsoft Corporation. All rights reserved.
5  //-----------------------------------------------------------------------------
6
7  #include "d3dx.h"
8  #include "resource.h"
9  #include "stdio.h"
10
11 #define NAME_OF_THE_APP "D3DX - Gravity"
12
13 #define RELEASENULL(pObject) if (pObject) {pObject->Release(); pObject = NULL;}
14
15 #define NUM_PLANETS         300
16 #define NUM_PARTICLES       1500
17 #define NUM_MATERIALS       25
18 #define NUM_SPHERES         25  //reused by planets
19 #define FULLSCREEN_WIDTH    640
20 #define FULLSCREEN_HEIGHT   480
21
22 #define Y_OFFSET(distance)       distance*(Random(2.0f)-1.0f)
23
24 typedef struct _SUN
25 {
26     D3DXVECTOR4 pos;
27     float distance;
28     ID3DXSimpleShape* pSphere;
29     DWORD dwMaterial;
30 } SUN;
31
32 typedef struct _PLANET
33 {
34     D3DXVECTOR4 pos;
35     float distance;
36     DWORD dwSphere;
37     DWORD dwMaterial;
38     BOOL bAttract;
39 } PLANET;
40
41 typedef struct _PARTICLE
42 {
43     D3DXVECTOR4 pos;
44     float distance;
45     DWORD dwMaterial;
46     BOOL bAttract;
47 } PARTICLE;
48
49 class CGravity
50 {
51 public:
52     CGravity();
53     ~CGravity();
54     void                    PauseDrawing();
55     void                    RestartDrawing();
56     void                    UpdateTime();
```

```
    void                    UnInit();
    HRESULT                 InitD3DX();
    HRESULT                 InitRenderer();
    HRESULT                 HandleModeChanges();
    void                    DestroySpheres();
    HRESULT                 GenerateSpheres();
    HRESULT                 Draw();
    void                    ApplyGravity(   float* pfDistance,
                                            D3DXVECTOR4* pPos,
                                            BOOL* pbAttract);

    BOOL                    m_bD3DXReady;
    BOOL                    m_bActive;
    BOOL                    m_bIsFullscreen;

    HWND                    m_hwndMain;
    RECT                    m_rWindowedRect;

    LPDIRECT3DDEVICE7       m_pD3DDev;
    LPDIRECT3D7             m_pD3D;
    LPDIRECTDRAW7           m_pDD;

    float                   m_fViewRot[3];
    float                   m_fSunRot[3];

    ID3DXContext*           m_pD3DX;

    SUN                     m_Sun;
    PLANET                  m_Planets[NUM_PLANETS];
    PARTICLE                m_Particles[NUM_PARTICLES];

    ID3DXMatrixStack*       m_pWorldStack;
    ID3DXMatrixStack*       m_pViewStack;

    D3DLIGHT7               m_LightOnSun;
    D3DLIGHT7               m_LightFromSun;
    D3DMATERIAL7            m_SunMaterial;
    D3DMATERIAL7            m_PlanetMaterials[NUM_MATERIALS];
    ID3DXSimpleShape*       m_Spheres[NUM_SPHERES];

    double                  m_dAbsoluteTime;
    double                  m_dElapsedTime;
    double                  m_dPeriod;
    LARGE_INTEGER           m_liLastTime;
};

CGravity* g_pGravity;

CGravity::CGravity()
{
    m_bD3DXReady        = FALSE;
    m_bIsFullscreen     = FALSE;
    m_pD3DDev           = NULL;
    m_pD3D              = NULL;
    m_pDD               = NULL;
    m_pD3DX             = NULL;
    m_fViewRot[0]       =  1.0f;
    m_fViewRot[1]       = -1.0f;
    m_fViewRot[2]       =  0.0f;
```

```
116     m_fSunRot[0]        =  0.0f;
117     m_fSunRot[1]        =  0.0f;
118     m_fSunRot[2]        =  0.0f;
119     m_pWorldStack       = NULL;
120     m_pViewStack        = NULL;
121     m_bActive           = !m_bIsFullscreen;
122     m_liLastTime.QuadPart   = 0;
123     m_dAbsoluteTime     = 0;
124     m_dElapsedTime      = 0;
125     m_Sun.pSphere       = NULL;
126
127     for( int i = 0; i < NUM_SPHERES; i++ )
128     {
129         m_Spheres[i] = NULL;
130     }
131
132     LARGE_INTEGER liFrequency;
133     if(!QueryPerformanceFrequency(&liFrequency))
134         liFrequency.QuadPart = 1;
135
136     m_dPeriod = (double)1/liFrequency.QuadPart;
137     if(!QueryPerformanceCounter(&m_liLastTime))
138         m_liLastTime.QuadPart = 0;
139 }
140
141 CGravity::~CGravity()
142 {
143     g_pGravity->UnInit();
144 }
145
146 void InterpretError(HRESULT hr)
147 {
148     char errStr[100];
149     D3DXGetErrorString(hr, 100, errStr );
150     MessageBox(NULL,errStr,"D3DX Error",MB_OK);
151 }
152
153 float Random(float fMax)
154 {
155     return fMax*rand()/RAND_MAX;
156 }
157
158 void CGravity::UpdateTime()
159 {
160     LARGE_INTEGER liCurrTime;
161     if(!QueryPerformanceCounter(&liCurrTime))
162         liCurrTime.QuadPart = m_liLastTime.QuadPart + 1;
163
164     m_dElapsedTime = (double)(liCurrTime.QuadPart - m_liLastTime.QuadPart)*
165                                 m_dPeriod;
166
167     m_dAbsoluteTime += m_dElapsedTime;
168     m_liLastTime = liCurrTime;
169 }
170
171
172 void CGravity::PauseDrawing()
173 {
174     g_pGravity->m_bActive = FALSE;
```

```
175     if( m_bIsFullscreen )
176         ShowCursor(TRUE);
177 }
178 
179 void CGravity::RestartDrawing()
180 {
181     g_pGravity->m_bActive = TRUE;
182     if( m_bIsFullscreen )
183         ShowCursor(FALSE);
184 }
185 
186 //-----------------------------------------------------------------------------
187 // Renderer Initialization Code
188 //-----------------------------------------------------------------------------
189 
190 HRESULT CGravity::InitD3DX()
191 {
192     HRESULT hr;
193     DWORD i;
194     char buff[1024];
195 
196     if( FAILED( hr = D3DXInitialize() ) )
197         return hr;
198 
199 
200 
201     // D3DX Initialization
202     hr = D3DXCreateContextEx(   D3DX_DEFAULT,           // D3DX handle
203                                 m_bIsFullscreen ? D3DX_CONTEXT_FULLSCREEN:0, // flags
204                                 m_hwndMain,
205                                 NULL,                   // focusWnd
206                                 D3DX_DEFAULT,           // colorbits
207                                 D3DX_DEFAULT,           // alphabits
208                                 D3DX_DEFAULT,           // numdepthbits
209                                 0,                      // numstencilbits
210                                 D3DX_DEFAULT,           // numbackbuffers
211                                 m_bIsFullscreen? FULLSCREEN_WIDTH:D3DX_DEFAULT,  //
    width
212                                 m_bIsFullscreen? FULLSCREEN_HEIGHT:D3DX_DEFAULT, //
    height
213                                 D3DX_DEFAULT,           // refresh rate
214                                 &m_pD3DX                // returned D3DX interface
215                         );
216     if( FAILED(hr) )
217         return hr;
218 
219     m_pD3DDev = m_pD3DX->GetD3DDevice();
220     if( m_pD3DDev == NULL )
221         return E_FAIL;
222 
223     m_pD3D = m_pD3DX->GetD3D();
224     if( m_pD3D == NULL )
225         return E_FAIL;
226 
227     m_pDD = m_pD3DX->GetDD();
228     if( m_pDD == NULL )
229         return E_FAIL;
230 
231     m_bD3DXReady = TRUE;
```

```
232     return InitRenderer();
233 }
234 
235 // ******************************************************************************
236 // Renderer Initialization Code
237 // ******************************************************************************
238 
239 HRESULT CGravity::InitRenderer()
240 {
241     HRESULT hr;
242     int i;
243 
244     if( !m_bD3DXReady )
245         return E_FAIL;
246 
247     hr = m_pD3DX->SetClearColor(D3DRGBA(0,0,0,0));
248     if( FAILED(hr) )
249         return hr;
250 
251     srand(4);
252     hr = m_pD3DDev->SetRenderState( D3DRENDERSTATE_DITHERENABLE, TRUE );
253     if ( FAILED(hr) )
254         return hr;
255 
256     hr = m_pD3DDev->SetRenderState( D3DRENDERSTATE_SPECULARENABLE, TRUE );
257     if ( FAILED(hr) )
258         return hr;
259 
260     hr = m_pD3DX->Clear(D3DCLEAR_TARGET|D3DCLEAR_ZBUFFER);
261     if ( FAILED(hr) )
262         return hr;
263 
264     D3DVALUE dvR,dvG,dvB;
265     dvR = 1.0f;
266     dvG = 0.9f;
267     dvB = 0.6f;
268     memset(&m_LightFromSun,0,sizeof(D3DLIGHT7));
269 
270     // Light which illuminates the "planets"
271     m_LightFromSun.dltType = D3DLIGHT_POINT;
272     m_LightFromSun.dvAttenuation0 = 0.5f;
273     m_LightFromSun.dvAttenuation1 = 0.008f;
274     m_LightFromSun.dvAttenuation2 = 0.0f;
275     m_LightFromSun.dcvDiffuse.dvR = dvR;
276     m_LightFromSun.dcvDiffuse.dvG = dvG;
277     m_LightFromSun.dcvDiffuse.dvB = dvB;
278     m_LightFromSun.dcvSpecular.dvR = dvR;
279     m_LightFromSun.dcvSpecular.dvG = dvG;
280     m_LightFromSun.dcvSpecular.dvB = dvB;
281     m_LightFromSun.dvRange = 5000.0f;
282 
283     // Light which illuminates the "sun"
284     memcpy(&m_LightOnSun, &m_LightFromSun, sizeof(D3DLIGHT7) );
285     m_LightOnSun.dvAttenuation0 = 0.0f;
286     m_LightOnSun.dvAttenuation1 = 0.003f;
287     m_LightOnSun.dvAttenuation2 = 0.0f;
288     m_LightOnSun.dcvDiffuse.dvR = dvR;
289     m_LightOnSun.dcvDiffuse.dvG = dvG;
290     m_LightOnSun.dcvDiffuse.dvB = dvB;
```

```
291     m_LightOnSun.dcvSpecular.dvR = 1.0f;
292     m_LightOnSun.dcvSpecular.dvG = 1.0f;
293     m_LightOnSun.dcvSpecular.dvB = 1.0f;
294 
295     hr = m_pD3DDev->LightEnable( 0, TRUE );
296     if ( FAILED(hr) )
297         return hr;
298 
299     memset(&m_SunMaterial,0,sizeof(D3DMATERIAL7));
300     m_SunMaterial.diffuse.r = dvR;
301     m_SunMaterial.diffuse.g = dvG;
302     m_SunMaterial.diffuse.b = dvB;
303     m_SunMaterial.specular.r = 1.0f;
304     m_SunMaterial.specular.g = 1.0f;
305     m_SunMaterial.specular.b = 1.0f;
306     m_SunMaterial.power = 3.0f;
307 
308     for( i = 0; i < NUM_MATERIALS; i++ )
309     {
310         memcpy(&m_PlanetMaterials[i],&m_SunMaterial,sizeof(D3DMATERIAL7));
311         m_PlanetMaterials[i].diffuse.r = Random(1.0f);
312         m_PlanetMaterials[i].diffuse.g = Random(1.0f);
313         m_PlanetMaterials[i].diffuse.b = Random(1.0f);
314         m_PlanetMaterials[i].power = 1.0f;
315     }
316 
317     if( FAILED( hr = GenerateSpheres() ) )
318         return hr;
319 
320     float fPlanetRad;
321     for( i = 0; i < NUM_PARTICLES; i++ )
322     {
323         // Make the planets.
324         fPlanetRad = (float)Random(3.2f) + 0.7f;
325 
326         D3DXMATRIX initRot;
327         D3DXMatrixRotationAxis(&initRot, &D3DXVECTOR3( 0.0f, 1.0f, 0.0f ),
Random(2*D3DX_PI) + 1 );
328         m_Particles[i].distance = Random(400.0f) + 50.0f;
329         D3DXVECTOR4 vUnit( 1.0f, Y_OFFSET(m_Particles[i].distance/5000.0f), 0.0f, 1.0f
);
330         D3DXVec4Normalize(&vUnit,&vUnit);
331         D3DXVec4Transform( &vUnit, &vUnit, &initRot );
332         m_Particles[i].pos = vUnit;
333 
334         m_Particles[i].dwMaterial = (DWORD)Random((FLOAT) NUM_MATERIALS);
335         m_Particles[i].bAttract = TRUE;
336     }
337 
338     // Create Matrix Stack
339     D3DXCreateMatrixStack( 0, &m_pWorldStack );
340 
341     // Create Matrix Stack
342     D3DXCreateMatrixStack( 0, &m_pViewStack );
343 
344     return S_OK;
345 }
346 
347 // ******************************************************************************
```

```
348 // GenerateSpheres
349 // ******************************************************************************
350
351 HRESULT CGravity::GenerateSpheres()
352 {
353     HRESULT hr;
354
355     float fPlanetRad;
356     for( int i = 0; i < NUM_SPHERES; i++ )
357     {
358         fPlanetRad = (float)Random(3.2f) + 0.7f;
359         hr = D3DXCreateSphere(  m_pD3DDev,
360                                 fPlanetRad,
361                                 (int)max(4,fPlanetRad*2),
362                                 (int)max(3,fPlanetRad*3),
363                                 1,
364                                 &m_Spheres[i] );
365         if( FAILED(hr) )
366             return hr;
367     }
368
369     // Make the sun.
370     hr = D3DXCreateSphere(m_pD3DDev,20.0f,50,25,1,&m_Sun.pSphere);
371     if( FAILED(hr) )
372         return hr;
373
374     for( i = 0; i < NUM_PLANETS; i++ )
375     {
376         // Make the planets.
377         fPlanetRad = (float)Random(3.2f) + 0.7f;
378
379         D3DXMATRIX initRot;
380         D3DXMatrixRotationAxis(&initRot, &D3DXVECTOR3( 0.0f, 1.0f, 0.0f ),
    Random(2*D3DX_PI) + 1 );
381         m_Planets[i].distance = Random(400.0f) + 50.0f;
382         D3DXVECTOR4 vUnit( 1.0f, Y_OFFSET((float) m_Planets[i].distance/5000.0f), 0.0f,
    0.0f );
383         D3DXVec4Normalize(&vUnit,&vUnit);
384         D3DXVec4Transform( &vUnit, &vUnit, &initRot );
385         m_Planets[i].pos = vUnit;
386
387
388         m_Planets[i].dwSphere = (DWORD)Random((FLOAT) NUM_SPHERES);
389
390         m_Planets[i].dwMaterial = (DWORD)Random((FLOAT) NUM_MATERIALS);
391         m_Planets[i].bAttract = TRUE;
392     }
393
394     return S_OK;
395 }
396
397 void CGravity::DestroySpheres()
398 {
399     for( int i = 0; i < NUM_SPHERES; i++ )
400     {
401         RELEASENULL(m_Spheres[i]);
402     }
403 }
404
```

```
405 void CGravity::UnInit()
406 {
407     DestroySpheres();
408     RELEASENULL(m_Sun.pSphere);
409     RELEASENULL(m_pD3DDev);
410     RELEASENULL(m_pD3D);
411     RELEASENULL(m_pDD);
412     RELEASENULL(m_pWorldStack);
413     RELEASENULL(m_pViewStack);
414     RELEASENULL(m_pD3DX);
415     m_bD3DXReady = FALSE;
416     D3DXUninitialize();
417 }
418 
419 void CGravity::ApplyGravity(float* pfDistance, D3DXVECTOR4* pPos, BOOL* pbAttract)
420 {
421     BOOL bFlippedState = FALSE;
422     if( *pbAttract == TRUE )
423     {
424         // Do some fake gravity stuff... (a little too much gravity... :) )
425         if( *pfDistance > 1 )
426         {
427             // Get sucked towards the sun:
428             *pfDistance *= (float)pow(0.999,m_dElapsedTime*10);
429 
430             if( *pfDistance < 50 )
431             {
432                 *pfDistance -= (float)(m_dElapsedTime*15);
433             }
434             else if( *pfDistance < 100 )
435             {
436                 *pfDistance -= (float)(m_dElapsedTime*10);
437             }
438             else if( *pfDistance < 150 )
439             {
440                 *pfDistance -= (float)(m_dElapsedTime*5);
441             }
442             *pfDistance = max(0.1f,*pfDistance);
443 
444             D3DXMATRIX rot;
445             D3DXMatrixRotationAxis(&rot, &D3DXVECTOR3( 0.0f, 1.0f, 0.0f ),
446                                    (float)(80.0f*pow(*pfDistance,-1.1f)
    *m_dElapsedTime));
447             D3DXVec4Transform( pPos, pPos, &rot );
448             (*pPos).y *= (float)pow(0.999,m_dElapsedTime*30);
449         }
450         if( *pfDistance <= 1 )
451         {
452             // Teleport out of the sun.
453             *pbAttract = FALSE;
454             bFlippedState = TRUE;
455         }
456     }
457     if( *pbAttract == FALSE )
458     {
459         if( *pfDistance > 500 )
460         {
461             // move to a new far away location, and
462             // start getting sucked in again
```

```
463             *pfDistance = Random(100.0f) + 500.0f;
464             (*pPos).y = Y_OFFSET(*pfDistance/10000.0f);
465             D3DXVec4Normalize(pPos,pPos);
466             *pbAttract = TRUE;
467             return;
468         }
469         if( bFlippedState )
470         {
471             (*pPos).y = 20.0f;
472             D3DXVec4Normalize(pPos,pPos);
473             if( Random(1.0f) > 0.5f )
474             {
475                 (*pPos).y = -(*pPos).y;
476             }
477             *pfDistance = 10.0f;
478         }
479         *pfDistance *= (float) pow(2.0, m_dElapsedTime);
480     }
481 }
482
483 // ******************************************************************************
484 // Rendering Code
485 // ******************************************************************************
486
487 HRESULT CGravity::Draw()
488 {
489     HRESULT hr;
490     int i;
491
492     if( !m_bD3DXReady )
493     {
494         return E_FAIL;
495     }
496     if( !m_bActive )
497     {
498         return S_OK;
499     }
500
501     hr = m_pD3DDev->BeginScene();
502     if ( SUCCEEDED(hr) )
503     {
504         hr = m_pD3DX->Clear(D3DCLEAR_TARGET|D3DCLEAR_ZBUFFER);
505         if ( FAILED(hr) )
506             return hr;
507
508         UpdateTime();
509
510         float fViewDist = 400+300*(float)sin(m_dAbsoluteTime*0.2);
511
512         m_LightOnSun.dvPosition.dvX = 0.0f;
513         m_LightOnSun.dvPosition.dvY = 0.0f;
514         m_LightOnSun.dvPosition.dvZ = -100.f + fViewDist;
515         m_pD3DDev->SetLight( 0, &m_LightOnSun );
516
517         // Set up state for drawing the sun.
518
519         m_fSunRot[0] = (float)(m_dAbsoluteTime/2);
520         m_fSunRot[1] = (float)(m_dAbsoluteTime/2);
521         m_fSunRot[2] = (float)(m_dAbsoluteTime);
```

```
522
523         m_pD3DDev->SetMaterial(&m_SunMaterial);
524
525         D3DXMATRIX matSunWorld, matTemp;
526         D3DXMatrixRotationAxis(&matSunWorld, &D3DXVECTOR3( 1.0f, 0.0f, 0.0f ),
527                                m_fSunRot[0] );
528         D3DXMatrixRotationAxis(&matTemp, &D3DXVECTOR3( 0.0f, 1.0f, 0.0f ),
529                                m_fSunRot[1] );
530         D3DXMatrixMultiply(&matSunWorld,&matSunWorld,&matTemp);
531         D3DXMatrixRotationAxis(&matTemp, &D3DXVECTOR3( 0.0f, 0.0f, 1.0f ),
532                                m_fSunRot[2] );
533         D3DXMatrixMultiply(&matSunWorld,&matSunWorld,&matTemp);
534         D3DXMatrixTranslation(&matTemp,0.0f,0.0f,fViewDist);
535         D3DXMatrixMultiply(&matSunWorld,&matSunWorld,&matTemp);
536
537         m_pD3DDev->SetTransform( D3DTRANSFORMSTATE_WORLD,
538                                  (D3DMATRIX *)matSunWorld );
539
540         D3DXMATRIX matSunView;
541         D3DXMatrixIdentity(&matSunView);
542         m_pD3DDev->SetTransform( D3DTRANSFORMSTATE_VIEW,
543                                  (D3DMATRIX *)matSunView );
544
545         m_Sun.pSphere->Draw();
546
547         m_pViewStack->LoadIdentity();
548         m_fViewRot[0]-=(float)(0.075*m_dElapsedTime);
549         m_fViewRot[1]+=(float)(0.01*m_dElapsedTime);
550         m_fViewRot[2]+=(float)(0.015*m_dElapsedTime);
551
552         m_pViewStack->RotateAxis( &D3DXVECTOR3( 1.0f, 0.0f, 0.0f ),
553                                   m_fViewRot[0] );
554         m_pViewStack->RotateAxis( &D3DXVECTOR3( 0.0f, 1.0f, 0.0f ),
555                                   m_fViewRot[1] );
556         m_pViewStack->RotateAxis( &D3DXVECTOR3( 0.0f, 0.0f, 1.0f ),
557                                   m_fViewRot[2] );
558         m_pViewStack->Translate(0.0f,0.0f,fViewDist);
559         m_pD3DDev->SetTransform( D3DTRANSFORMSTATE_VIEW,
560                                  (D3DMATRIX*)m_pViewStack->GetTop() );
561
562
563         // Set up state for drawing the planets
564
565         m_pD3DDev->SetLight( 0, &m_LightFromSun );
566
567         // Draw the planets
568         for( i = 0; i < NUM_PLANETS; i++ )
569         {
570             m_pD3DDev->SetMaterial(&m_PlanetMaterials[m_Planets[i].dwMaterial]);
571             m_pWorldStack->LoadIdentity();
572
573             // Do some fake gravity stuff... (a little too much gravity... :) )
574             ApplyGravity(&m_Planets[i].distance,&m_Planets[i].pos, &m_Planets[i]
    .bAttract);
575             m_pWorldStack->Translate( m_Planets[i].pos.x * m_Planets[i].distance,
576                                       m_Planets[i].pos.y * m_Planets[i].distance,
577                                       m_Planets[i].pos.z * m_Planets[i].distance );
578
579             m_pD3DDev->SetTransform( D3DTRANSFORMSTATE_WORLD,
```

```
                                    (D3DMATRIX *)m_pWorldStack->GetTop() );

            m_Spheres[m_Planets[i].dwSphere]->Draw();
        }

        m_pWorldStack->LoadIdentity();
        m_pD3DDev->SetTransform( D3DTRANSFORMSTATE_WORLD,
                                 (D3DMATRIX *)m_pWorldStack->GetTop() );

        // Draw the particles
        D3DVERTEX vParticle;
        for( i = 0; i < NUM_PARTICLES; i++ )
        {
            m_pD3DDev->SetMaterial(&m_PlanetMaterials[m_Particles[i].dwMaterial]);

            // Do some fake gravity stuff... (a little too much gravity... :) )
            ApplyGravity(&m_Particles[i].distance,&m_Particles[i].pos,
                         &m_Particles[i].bAttract);

            vParticle.dvX = m_Particles[i].pos.x* m_Particles[i].distance;
            vParticle.dvY = m_Particles[i].pos.y* m_Particles[i].distance;
            vParticle.dvZ = m_Particles[i].pos.z* m_Particles[i].distance;
            vParticle.dvNX = -m_Particles[i].pos.x;
            vParticle.dvNY = -m_Particles[i].pos.y;
            vParticle.dvNZ = -m_Particles[i].pos.z;
            m_pD3DDev->DrawPrimitive(D3DPT_POINTLIST,
                                     D3DFVF_VERTEX,
                                     &vParticle,
                                     1,
                                     D3DDP_WAIT );
        }



        m_pD3DDev->EndScene();
    }

    hr = m_pD3DX->UpdateFrame( 0 );
    if ( hr == DDERR_SURFACELOST || hr == DDERR_SURFACEBUSY )
        hr = HandleModeChanges();

    return hr;
}

HRESULT CGravity::HandleModeChanges()
{
    HRESULT hr;
    hr = m_pDD->TestCooperativeLevel();

    if( SUCCEEDED( hr ) || hr == DDERR_WRONGMODE )
    {
        UnInit();

        if(FAILED(hr = InitD3DX()))
            return hr;
    }
    else if( hr != DDERR_EXCLUSIVEMODEALREADYSET &&
             hr != DDERR_NOEXCLUSIVEMODE )
    {
```

```
639         // Busted!!
640         return hr;
641     }
642     return S_OK;
643 }
644 
645 // ******************************************************************************
646 // Windowing Code...
647 // ******************************************************************************
648 
649 LRESULT CALLBACK WndProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
650 {
651     switch(uMsg)
652     {
653     case WM_ACTIVATEAPP:
654         {
655             if( !g_pGravity )
656                 break;
657 
658             if( g_pGravity->m_bIsFullscreen )
659             {
660                 if( (BOOL)wParam )
661                     g_pGravity->RestartDrawing();
662                 else
663                     g_pGravity->PauseDrawing();
664             }
665         }
666         break;
667     case WM_CREATE:
668         break;
669     case WM_CLOSE:
670         PostQuitMessage(0);
671         break;
672     case WM_SIZE:
673         if( g_pGravity
674             && g_pGravity->m_bD3DXReady
675             && !g_pGravity->m_bIsFullscreen
676             )
677         {
678             HRESULT hr;
679 
680             if( wParam == SIZE_MINIMIZED )
681             {
682                 g_pGravity->m_bActive = FALSE;
683                 break;
684             }
685             else if( LOWORD(lParam)>0 && HIWORD(lParam)>0 )
686             {
687                 if( FAILED(hr = g_pGravity->m_pD3DX->Resize(LOWORD(lParam),
HIWORD(lParam))))
688                 {
689                     InterpretError(hr);
690                     g_pGravity->m_bD3DXReady = FALSE;
691                     PostQuitMessage(0);
692                 }
693             }
694             g_pGravity->m_bActive = TRUE;
695 
696         }
```

```
697         break;
698      case WM_KEYDOWN:
699         switch( wParam )
700         {
701         case VK_ESCAPE:
702         {
703             PostQuitMessage(0);
704             break;
705         }
706         }
707         break;
708     case WM_COMMAND:
709         if ( 1 == HIWORD(wParam) )
710         {
711             switch ( LOWORD(wParam) )
712             {
713             case IDM_FULLSCREEN:
714                 if( g_pGravity && g_pGravity->m_bD3DXReady )
715                 {
716                     HRESULT hr;
717                     g_pGravity->m_bIsFullscreen = ! g_pGravity->m_bIsFullscreen;
718                     g_pGravity->m_bD3DXReady = FALSE;
719 
720                     if ( g_pGravity->m_bIsFullscreen )
721                     {
722                         // going to fullscreen
723                         GetWindowRect( hwnd, &g_pGravity->m_rWindowedRect );
724                     }
725                     ShowCursor(!(g_pGravity->m_bIsFullscreen));
726                     hr = g_pGravity->m_pD3DX->Clear(D3DCLEAR_TARGET|D3DCLEAR_ZBUFFER);
727                     if ( FAILED(hr) )
728                     {
729                         InterpretError(hr);
730                         g_pGravity->PauseDrawing();
731                         PostQuitMessage(-1);
732                         break;
733                     }
734                     g_pGravity->UnInit();
735 
736                     if ( !g_pGravity->m_bIsFullscreen )
737                     {
738                         RECT& r = g_pGravity->m_rWindowedRect;
739                         SetWindowPos(hwnd, HWND_NOTOPMOST,
740                                     r.left,
741                                     r.top,
742                                     r.right-r.left,
743                                     r.bottom-r.top,
744                                     SWP_NOACTIVATE );
745                     }
746 
747                     hr = g_pGravity->InitD3DX();
748                     if ( FAILED(hr) )
749                     {
750                         InterpretError(hr);
751                         g_pGravity->PauseDrawing();
752                         PostQuitMessage(-1);
753                         break;
754                     }
755                 }
```

```
                break;
            }
        }
        break;
    default:
        break;
    }

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

}

int PASCAL WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
    LPSTR lpszCmdLine, int nCmdShow)
{
    HRESULT     hr;
    MSG         msg;
    WNDCLASS    wc;
    HACCEL      hAccelApp;
    HCURSOR     hcur = NULL;
    int         ret = 0;

    g_pGravity = new CGravity; // set up our data AFTER starting up d3dx!
    if( !g_pGravity )
    {
        ret = -1;
        goto Exit;
    }

    // Register the window class for the main window.

    if (!hPrevInstance)
    {
        hcur = CopyCursor(LoadCursor(NULL, IDC_ARROW));

        wc.style = 0;
        wc.lpfnWndProc = (WNDPROC) WndProc;
        wc.cbClsExtra = 0;
        wc.cbWndExtra = 0;
        wc.hInstance = hInstance;
        wc.hIcon = LoadIcon(hInstance, MAKEINTRESOURCE(IDI_APP_ICON));
        wc.hCursor = hcur;
        wc.hbrBackground = (HBRUSH)GetStockObject(BLACK_BRUSH);
        wc.lpszMenuName =  NULL;
        wc.lpszClassName = NAME_OF_THE_APP;

        if (!RegisterClass(&wc))
        {
            ret = -1;
            goto Exit;
        }
    }

    // Create the window

    g_pGravity->m_hwndMain = CreateWindow(  NAME_OF_THE_APP,
                                            NAME_OF_THE_APP,
                                            WS_OVERLAPPEDWINDOW,
                                            CW_USEDEFAULT,
```

```
                                        CW_USEDEFAULT,
                                        400,
                                        400,
                                        (HWND) NULL,
                                        (HMENU) NULL,
                                        hInstance,
                                        (LPVOID) NULL);

    if (!g_pGravity->m_hwndMain)
    {
        ret = -1;
        goto Exit;
    }


    // Hide the cursor if necessary
    if( g_pGravity->m_bIsFullscreen )
    {
        ShowCursor(FALSE);
    }

    // Show the window
    ShowWindow(g_pGravity->m_hwndMain, nCmdShow);
    UpdateWindow(g_pGravity->m_hwndMain);

    hAccelApp = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDR_APP_ACCELERATOR));
    if ( !hAccelApp )
    {
        ret = -1;
        goto Exit;
    }

    // Initialize D3DX
    hr = g_pGravity->InitD3DX();
    if ( FAILED(hr) )
    {
        InterpretError(hr);
        ret = -1;
        goto Exit;
    }

    BOOL bGotMsg;
    PeekMessage( &msg, NULL, 0U, 0U, PM_NOREMOVE );
    while( WM_QUIT != msg.message  )
    {
        bGotMsg = PeekMessage( &msg, NULL, 0U, 0U, PM_REMOVE );

        if( bGotMsg )
        {
            if ( !TranslateAccelerator( g_pGravity->m_hwndMain, hAccelApp, &msg ) )
            {
                TranslateMessage( &msg );
                DispatchMessage( &msg );
            }
         }
        else
        {
            if( g_pGravity && g_pGravity->m_bActive )
            {
```

```
874                 hr = g_pGravity->Draw();
875                 if( FAILED(hr) )
876                 {
877                     InterpretError(hr);
878                     g_pGravity->m_bD3DXReady = FALSE;
879                     PostQuitMessage(-1);
880                 }
881             }
882             else
883             {
884                 WaitMessage();
885             }
886         }
887     }
888     delete g_pGravity; // clean up our data BEFORE shutting down d3dx!
889 
890 Exit:
891     if(hcur)
892         DestroyCursor(hcur);
893 
894     return ret;
895 }
```