30 Dakikada OpenGL'e Giriş

Yazan:
Ziver Alen Malhasoğlu
<malhasoglu (at) itu.edu.tr>

Aralık 2002

Özet

Bu belge OpenGL'i tanıtmak amacıyla hazırlanmıştır. OpenGL'i duymuş ama bu konuda bir fikri olmayanlara bilgi vermek amacındadır.

Belge kısaca Linux'da nasıl OpenGL programlama yapılacağını anlatır, örnek birkaç program göstererek elde edilebilecek sonuçlar hakkında bilgi verir ve daha fazla bilgi edinilebilecek kaynakları gösterir.

Bu belgenin özgün sürümünü `http://www.students.itu.edu.tr/~malhasoglu/OpenGL-«GirisDok/` adresinde bulabilirsiniz.

Konu Başlıkları

Geçmiş

1.0	5 Aralık 2002	ZAM
İlk sürüm		

Yasal Uyarı

1. OpenGL Programlamaya Giriş

1.1. API (Uygulama Programlama Arayüzü) Nedir?

Bir işletim sisteminin veya bir programın sunduğu bir hizmeti kullanmak için verilmiş yöntemlerdir. Bunu bir kitaplıktan işlev çağırmak gibi düşünebiliriz. Biz isteği API ile yaparız ve isteğimiz yerine getirilir, sonra programımız normal çalışmasına devam eder.

1.2. OpenGL Nedir?

Kitaplıktır

Open Graphics Library (OpenGL), 2 veya 3 boyutlu grafik çizdirmek için kullanılan bir API'dir bir başka deyişle kitaplıktır. Bu kitaplık yazılım geliştiricilere grafik donanımını yönetme şansı verir.

Taşınabilirdir

Bu kitaplık işletim sisteminden ve işletim sisteminin çalıştığı platformdan bağımsızdır. Nasıl ki ekrana yazı yazmak kullanıcıdan veri almak ANSI C'de `printf()` ve scanf() gibi işlevlerle standartlaştırılmış ve hangi işletim sistemine giderseniz gidin bu iki işlev aynı işi yapıyorsa, OpenGL kitaplığı da ekrana grafik çizmeyi standartlaştırmıştır. OpenGL sayesinde grafik kartının modeli veya işlemcinin mimarisi gibi donanımsal etkenlerden bağımsız programlama yapılır. Ayrıca aynı zamanda işletim sisteminden de bağımsız programlama yapılır. Kolay kullanım ve bu "taşınabilirlik" özellikleri yüzünden OpenGL popüler bir araç olmuştur.

İşletim sisteminden bağımsızdır

OpenGL kullanan bir programı işletim sisteminizde çalıştırmanız için öncelikle işletim sisteminizde programın çalışırken kullanacağı işlevleri içeren kitaplığın bulunması gerekir, bu kitaplıkların genel adı "runtime–library", türkçesi çalışma anı kitaplığıdır.

OpenGL çalışma anı kitaplığı Linux, Unix, Mac OS, OS/2, Windows 95/98/NT/2000, OPENStep ve BeOS işletim sistemlerinde vardır. Windows ailesinde standart olarak gelir. Yani OpenGL kullanan bir programı çalıştırmak için bir çaba harcamazsınız.

Pencere yöneticisinden bağımsızdır

OpenGL kullanılarak yazılmış programlar, Win32, MacOS ve X–Window pencere yöneticilerinde sorunsuz çalışırlar.

Birçok programlama dilinden kullanılabilir

Ada, C, C++, C# (SharpGL adı verilen sınıflar sayesinde), Fortran, Python,Perl ve Java programlama dilleri kullanılarak OpenGL kitaplığından faydalanılabilir. [*#2*]

1.3. GLUT Hakkında

Taşınabilirlikten bahsetmişken GLUT'tan sözetmemek olmaz. OpenGL platformdan bağımsız olduğu için bazı işlemler bu kitaplık ile yapılamaz. Örneğin kullanıcıdan veri almak, bir pencere çizdirmek gibi işler hep kullanılan pencere yöneticisi ve işletim sistemine bağlıdır. Bu yüzden bir an için OpenGL'in bu durumlarda platforma bağımlı olduğunu düşünebiliriz. Çünkü penceresini her pencere yöneticisinde farklı çizdirecek bir canlandırma programı yazmak demek her bilgisayarda çalışacak ayrı pencere açma kodu yazmak demektir. Bu ise OpenGL'in doğasına aykırıdır. Neyse ki OpenGL Araç Kiti (GLUT – OpenGL Utility Toolkit) var da yazılım geliştiricileri bir platforma bağlılıktan tamamen kurtarıyor.

GLUT, birçok işletim sistemine aktarılmış bir kitaplıktır. Amacı OpenGL programlarının pencerelerini oluşturmak, klavye ve fareden veri almak gibi ihtiyaçlarını karşılamaktır.

GLUT olmadan da OpenGL programlama yapılabilir, örneğin Linux'ta kullanılan X–Window sistemin kendi işlevleri kullanılarak pencere çizdirilebilir fakat bu kod sadece X–Window'da çalışır. Kod Windows'a götürülüp derlendiğinde çalışmaz, çünkü Windows'da X–Window işlevleri yoktur!

Bu yüzden bu belgede GLUT kitaplığı kullanılarak klavye ve fare için işletim sisteminden bağımsız giris/çıkış işlemleri yapılması sağlanmıştır.

2. Ne gerekir? Nasıl yapılır?

2.1. OpenGL ile programlamaya başlamak: Mesa kurulumu

Bu belgede OpenGL ile programlama konusunu anlatmak için C programlama dili ve Linux işletim sistemi seçilmiştir.

OpenGL programlarını çalıştırmanız için gereken kitaplık bilgisayarınızda yüklü olabilir fakat sadece bu kitaplıkların varlığı OpenGL kullanan programlar geliştirmenize yetmez. Programınızda kullanacağınız başlık dosyalarını ve geliştirme ortamınızın kullandığı iç bağlantıların ihtiyaç duyacağı OpenGL kitaplık dosyalarına da ihtiyacınız olacaktır.

OpenGL kitaplığındaki birçok işlevi kapsayan ve amacı OpenGL'i uyarlamak olmasa da OpenGL kullanan birçok programı çalıştırabilen "Açık Kod" lisansına sahip bir kitaplık mevcuttur; Mesa. Mesa'nın bu belge yazıldığı sırada son kararlı versiyonu 5.0 idi; bu kitaplığı `http://www.mesa3d.org` adresinden indirdikten ve kurduktan sonra OpenGL işlevlerini kullanmaya başlayabilirsiniz. Yeri gelmişken Mesa'nın, Windows'da derlenip kullanılabileceğini belirtelim. Mesa'yı Linux dağıtımınızaki RPM paketlerinden veya en son sürümünü Mesa'nın resmi sitesinden indirip derleyerek elde edebilirsiniz. Burada kaynak koddan derleme anlatılacaktır.

`http://www.mesa3d.org` sayfasından `MesaLib` ve `MesaDemos` dosyalarını indirin ve `root` olarak şu komutları verin (`MesaLib.tar.gz` ve `MesaDemos.tar.gz` dosyalarını indirdiğiniz varsayılmıştır):

```
# tar xzfv MesaLib.tar.gz
# tar xzfv MesaDemos.tar.gz
# cd Mesa-5.0/
# ./configure
# make
# make install
```

Bu komutları verdikten sonra sorun (ki bende çıkmamıştı ;–), çıkarsa Mesa'nın sitesindeki SSS/Eposta Listesi gibi yardım kaynaklarına başvurunuz) sisteminizde kurulu ve kullanıma hazır bir OpenGL kitaplığınız olacaktır.

`make check` komutunu da verirseniz Mesa ile gelen örnek programlar da derlenecektir. Daha sonra bu demoları inceleyip nasıl çalıştıklarını öğrenebilirsiniz.

2.2. İlk OpenGL Programınız

Bir metin düzenleyici kullanarak glilk.c adında bir dosya oluşuralım ve içine şunları yazalım:

```
#include <GL/glut.h>

void myDisplay()
{
        glClear(GL_COLOR_BUFFER_BIT);
        glBegin(GL_POLYGON);
        glVertex2f(0.0, 0.0);
        glVertex2f(0.2, 0.0);
        glVertex2f(0.2, 0.5);
```

```
        glEnd();

        glFlush();
}

int main (int argc, char ** argv)
{
        glutInit (&argc, argv);
        glutCreateWindow("İşte ilk pencerem!");
        glutDisplayFunc(myDisplay);
        glutMainLoop();
        return(0);
}
```

Programı şu şekilde derleyebilirsiniz:

```
gcc glilk.c -o glilk -lGL -lGLU -lglut
```

Çalıştırmak için `./glilk` komutu kullanılabilir.

3. Bazı OpenGL İşlevlerinin Tanıtımı

Burada işlevler hakkında kısa bir bilgi verilmiştir. Daha ayrıntılı bilgi almak isterseniz man sayfalarına bakınız.

3.1. Temel İşlevler

```
void glutInit(int   *argcp,                                          işlev
              char **argv)
```

GLUT kitaplığının ilklendirilmesi için çağrılması gereken işlevdir. Bu çağırılmadan GLUT kitaplığından başka bir işlev çağrılamaz. Argümanlar programın `main()` işlevinden alınır ve değiştirilmeden kullanılır. Kullanımı söyledir:

```
glutInit(&argc, argv); // main(int argc, char ** argv) olduğu varsayılmıştır.
```

```
int glutCreateWindow(char isim)                                      işlev
```

Pencere sisteminde bir pencere oluşturur ve pencere başlığına *isim* parametresindeki metni yazar. Pencere numarası ile döner.

```
void glutDisplayFunc(int   (*islev) (void))                          işlev
```

Pencere ekrana çizildikten sonra pencerenin içerisinde gösterilecekleri oluşturan işlevin belirtileceği işlevdir (callback).

```
void glutMainLoop()                                                  işlev
```

Bu işlev sayesinde program kendisine gelecek olayları (event) dinlemeye başlar ve eğer tanımlıysa gelen olaya göre tanımlanmış bir işlevi (callback) çalıştırır.

```
void glClear(GLbitfield maske)                                       işlev
```

Tamponların içeriğini `glClearColor`, `glClearIndex`, `glClearDepth`, `glClearStencil` ve `glClearAccum` işlevleri ile belirlenen değerlerle temizler.

Parametre olarak GL_COLOR_BUFFER_BIT, GL_DEPTH_BUFFER_BIT, GL_ACCUM_BUFFER_BIT, GL_STENCIL_BUFFER_BIT sabitlerinden birini ya da bunların VEYAlanmış değerini alır.

```
void glBegin(enum kip)                                                    işlev
```

Bu işlev bir çizimin başlatıldığını belirtir. Aldığı parametre ise çizilen şeyin noktalar, çizgiler veya içi dolu çizgiler şeklinde görüneceğini belirtir.

Parametre olarak aldığı sembolik sabitler:

GL_POINTS
: Verilen noktaları nokta olarak çizer.

GL_LINES
: Verilen noktaları doğrularla birleştirir.

GL_POLYGON
: Verilen noktaları doğrularla birleştirir ve oluşan şeklin içini renklendirir.

GL_QUADS
: Verilen dört noktadan içi boyanmış dörtgen oluşturur.

GL_TRIANGLES
: Verilen üç noktadan içi boyanmış üçgen oluşturur.

GL_TRIANGLE_STRIP
: Şu noktalar glBegin ve glEnd arasında çizdirilmiş olsun: p0, p1, p2, p3, p4, p5. p0, p1 ve p2'den bir üçgen oluşturulur ve sonraki her nokta için önceki iki nokta birleştirilerek bir üçgen daha oluşturulur. Yani p3 ile p1 ve p2 birleştirilir. Daha sonra p4 ile p3 ve p2, ... vs.

GL_QUAD_STRIP
: GL_TRIANGLE_STRIP gibi çalışır, ama bu sefer verilen iki noktayı önceki iki nokta ile birleştirerek bir dörtgen oluşturur.

GL_TRIANGLE_FAN
: p0, p1, p2, p3, p4, p5 verilmiş olsun. p0, p1 ve p2 üçgeni çizilir. Daha sonra p4 için p0 ve p3 birleştirilerek yeni üçgen elde edilir. p5 ile p4 ve p0 birleştirilir, ...vs. Böylece şekil yelpaze gibi olur.

```
void glEnd()                                                              işlev
```

glBegin() ile başlayan çizim işleminin bittiğini belirtir. Çizdirilen şekil ekrana yazılmak üzere saklanır; glFlush() ile ekrana yazılır.

```
void glFlush()                                                            işlev
```

Eğer çizilenler tamponlanmışsa, tampon bellekteki tüm şekillerin ekrana basılmasını sağlar.

Sadece yukarıdaki işlevleri kullanarak basit geometrik şekillerin iki boyutta çizimi mümkündür. Yukarıdaki glilk.c adlı program bu işlevlerden yararlanarak yazdığımız ilk OpenGL uygulamamızdır.

```
void glClearColor(GLclampf kırmızı,                                       işlev
                  GLclampf yeşil,
                  GLclampf mavi,
                  GLclampf donukluk)
```

`glClear()` işleviyle temizlenen ekran renk tamponunun ne renk alacağını belirler. Her parametre 0 ya da 1 değerini alır. Örneğin *donukluk* değeri 0 ise şeffaflık, 1 ise donukluk elde edilir. Öntanımlı değerlerin tümü 0'dır.

```
void glColor3s(short kırmızı,                                    işlev
               short yeşil,
               short mavi)
```

Çizilecek şeklin rengini belirler. Öntanımlı değerlerin tümü 0'dır.

```
void glutInitWindowSize(int genişlik,                            işlev
                        int yükseklik)
```

Oluşturulan pencerenin boyutlarını belirler. Pencere boyutlarının öntanımlı değeri (300, 300)'dür.

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

Pencerenin konumlanacağı yeri belirtir. Pencere yerinin öntanımlı değeri (–1, –1)'dir, böylece yerini pencere yöneticisi belirler.

```
void glLineWidth(float genişlik)                                 işlev
```

Çizginin kalınlığını belirtir. Bu işlevle değiştirilmedikçe kalınlığın öntanımlı değeri 1.0'dır.

```
void glLineStipple(int   çarpan,                                 işlev
                   short örüntü)
```

Çizginin nokta nokta ya da düz çizgi şeklinde görünmesini ayarlar. Eğer *örüntü*deki bit 0 ise bu bite karşı gelen benek ekrana basılmaz, eğer 1 ise ekrana basılır; Böylece kesik kesik çizgi çizilebilir.

Örnek:

```
glLineStipple(3, 0xcccc); /* 0xCCCC = 1100110011001100 */
```

Bu örnekte ikilik *örüntü* 3 ile genişletilmiştir. Bu işlevle, ekrandaki çizgiler 6 beneklik gruplara ayrılacak ve bir grup ekrana basılacak, bir grup basılmayacaktır.

```
void glEnable()                                                  işlev
void glDisable()                                                 işlev
```

Performans artışı sağlamak için OpenGL'deki kesiklililik, ışıklandırma, kaplama gibi özellikler `glDisable()` ile kapatılabilir. Böylece bu özellikler şeklin ekranda oluşturulması sırasında gözardı edilecek şekil daha hızlı ortaya çıkacaktır.

`glEnable()` özelliğin kullanılmasını sağlarken `glDisable()` kullanılmaz hale getirir. Örneğin:

```
glEnable(GL_LINE_STIPPLE); // kesikli çizgi çizebilmek için
glEnable(GL_SMOOTH);       // renk geçişlerini yumuşatmak için
```

```
void glRecti(int x1,                                             işlev
             int y1,
             int x2,
             int y2)
```

İşlev, bir köşesi ilk iki parametresi ile çapraz köşesi ise son iki parametresi ile belirtilen bir dörtgenin çizilmesini sağlar.

İşlevin `glRects, glRectf, glRectd` türevleri de vardır. Tek farkları parametrelerinin sırasıyla `short, float`, ve `double` olmasıdır.

3.2. Dönüşümler

```
void glRotate(double açı,                                                    işlev
              double x,
              double y,
              double z)
```

Şekil, *açı* derece kadar koordinatları *x*, *y*, *z* ile belirtilen noktanın etrafında döndürülür.

```
void glTranslated(double x,                                                  işlev
                  double y,
                  double z)
```

Koordinatları *x*, *y*, *z* ile belirtilen noktaya koordinat sistemini öteler.

```
void gluOrtho2D(double sol,                                                  işlev
                double sağ,
                double alt,
                double üst)
```

İki boyutlu görüş peneresinin (clipping window) büyüklüğünü belirler.

```
void glLoadIdentity()                                                        işlev
```

Yapılmış tüm dönüşümlerin geri alınmasını sağlar.

```
void glScaled(double x,                                                      işlev
              double y,
              double z)
```

Bu dönüşüm sayesinde ölçekleme yapılır. Eğer girilen değerler 1'den küçükse nesneler küçültülür, 1'den büyükse nesneler büyütülür. Bu işlevin `float` parametreler alan `glScalef` isimli türevi de mevcuttur.

3.3. Olay Tanımlama İşlevleri

```
void glutReshapeFunc(void (*işlev) (int    genişlik,                         işlev
                                    double yükseklik)
```

Eğer pencere yeniden boyutlandırılırsa bu işlevin parametresi olan işlev çağrılır ve parametre olarak yeni genişlik ve yükseklik değerleri atanır.

```
void glutIdleFunc(void (*işlev) (void)                                       işlev
```

Hiçbir olay oluşturulmadığında çalıştırılacak işlevi belirtir.

3.4. Artalanda Tamponlama

Her seferinde ekrandaki görüntünün tazelenmesi CRT (=Cathode Ray Tube) tarafından yapılır. Bu olaya programımız müdahale edemez. Bellekte OpenGL'in çizim için kullandığı ekran bölgesine nokta eklemek çizgi çizmek gibi işlemlerle değişiklik yaptıkça ekrana bu değişiklikler anında yansıtılır. Ama tüm değişikliklerin yapıldıktan sonra ekranın tazelenmesini sağlayacak bir yol da vardır.

Çizimi başka bir bellek bölgesinde yapıp sonra ekran bölgesine aktarabiliriz (double buffering). Arka planda ekranın yeni görüntüsü işlevlerle hazırlanır ve bu sırada ön planda yani kullanıcının gördüğü pencerede bir

değişiklik olmaz. Biz programımızda `glFlush()` yerine `glSwapBuffers()` işlevini kullanırsak arka planda hazırladığımız değişiklikler ön plana yansır.

Bu durumu "ya hep ya hiç" şeklinde işlenmesi gereken verilere benzetebiliriz. Arka planda şeklin tamamı çizildikten sonra ön plana aktarılır. Böylece ekran tazelenmesi sırasında ekranın titrememesi sağlanır.

Ama artalanda tamponlama yapabilmek için penceremiz oluşturulmadan şu şekilde ilklendirme yapılması gerekmektedir:

```
glutInitDisplayMode (GLUT_RGBA | GLUT_DOUBLE);
```

`GLUT_DOUBLE` sayesinde çift tamponlu bir penceremiz olur. Artık `glFlush()` yerine `glSwapBuffers()` kullanılarak artalanda tamponlama yapılabilir.

3.5. Klavye ve Fare Kullanımı

3.5.1. Klavye İşlevleri

```
void glutKeyboardFunc(void (*işlev) (char tuş,
                                     int  x,
                                     int  y)
```
işlev

Klavyeden bir tuşa basıldığında bu işlev çağrılır. *x* ve *y* farenin o andaki konumunu belirtir. *tuş* ise klavyede basılan tuşu belirtir.

```
void glutSpecialFunc(void (*işlev) (int tuş,
                                    int x,
                                    int y)
```
işlev

Klavyedeki "F tuslarini" yani işlev tuşları için bu işlev kullanılır. Örnek:

```
  if(key == GLUT_KEY_F1){ printf("F1'e bastiniz.\n"); }
  else if(key == GLUT_KEY_UP) { printf("Yukari ok tusuna bastiniz.\n"); }
```

```
int glutGetModifiers()
```
işlev

Herhangi bir tuşa basılmışken CTRL, ALT veya SHIFT tuşlarından birine basılıp basılmadığını bu işlev sayesinde öğrenebiliriz. Örneğin `glutSpecialFunc` işlevi tarafından çalıştırılan `myFunction` işlevi şöyle bir kod içerebilir:

```
void myFunction(void){
  int modifier;
  ...
  modifier = glutGetModifiers();
  if(modifier = GLUT_ACTIVE_SHIFT){ printf("SHIFT tusuna bastiniz."); }
  ...
}
```

İşlevden dönen değeri `GLUT_ACTIVE_SHIFT`, `GLUT_ACTIVE_CTRL`, `GLUT_ACTIVE_ALT` sabitleri ile bulabiliriz.

3.5.2. Fare İşlevleri

```
void glutMouseFunc(void (*işlev) (int tuş,
                                  int durum,
                                  int x,
                                  int y)
```
işlev

Bu işlev farenin herhangi bir tuşuna basıldı veya bırakıldığı zaman calışır. *x* ve *y* farenin o andaki konumunu belirtir. *tuş* GLUT_LEFT_BUTTON, GLUT_RIGHT_BUTTON, GLUT_MIDDLE_BUTTON olarak tanımlanan sırasıyla sol, sağ ve orta fare tuşlarını belirtir. *durum* ise tuşun ne durumda olduğunu söyler. GLUT_UP, GLUT_DOWN sabitleriyle tanımlanır.

```
void glutMouseFunc(void (*işlev) (int x,                                    işlev
                                  int y)
```

Tuş basılı olarak fare hareket ettiğinde çalışan işlevdir. Farenin o anki konumu *x* ve *y* parametrelerine atanır.

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

Herhangi bir tuşa basılmaksızın farenin hareket edişi sırasında çağrılan işlevdir. Farenin o anki konumu *x* ve *y* parametrelerine atanır.

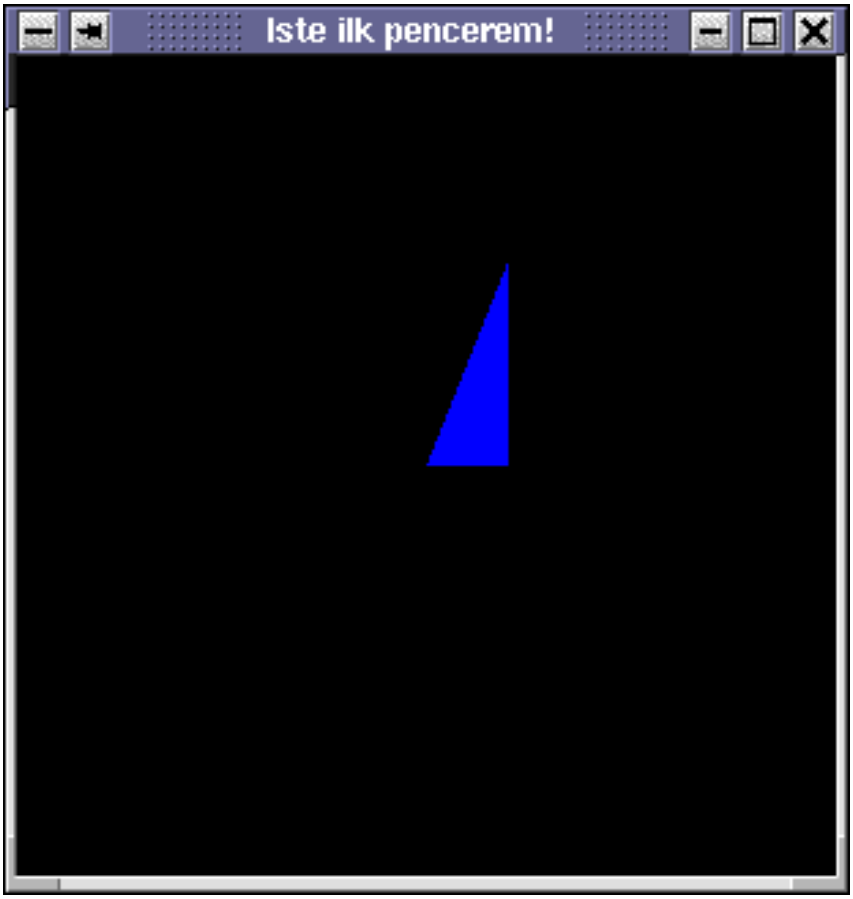
```
void glutEntryFunc(void (*işlev) (int durum)                                işlev
```

Fare pencere sınırlarına girince veya sınırlarından çıkınca çağrılan işlevdir. *durum* parametresinin aldığı değere göre GLUT_ENTERED ile farenin pencereye girdiği, GLUT_LEFT ile farenin pencereden çıktığı anlaşılır.

4. Örnekler

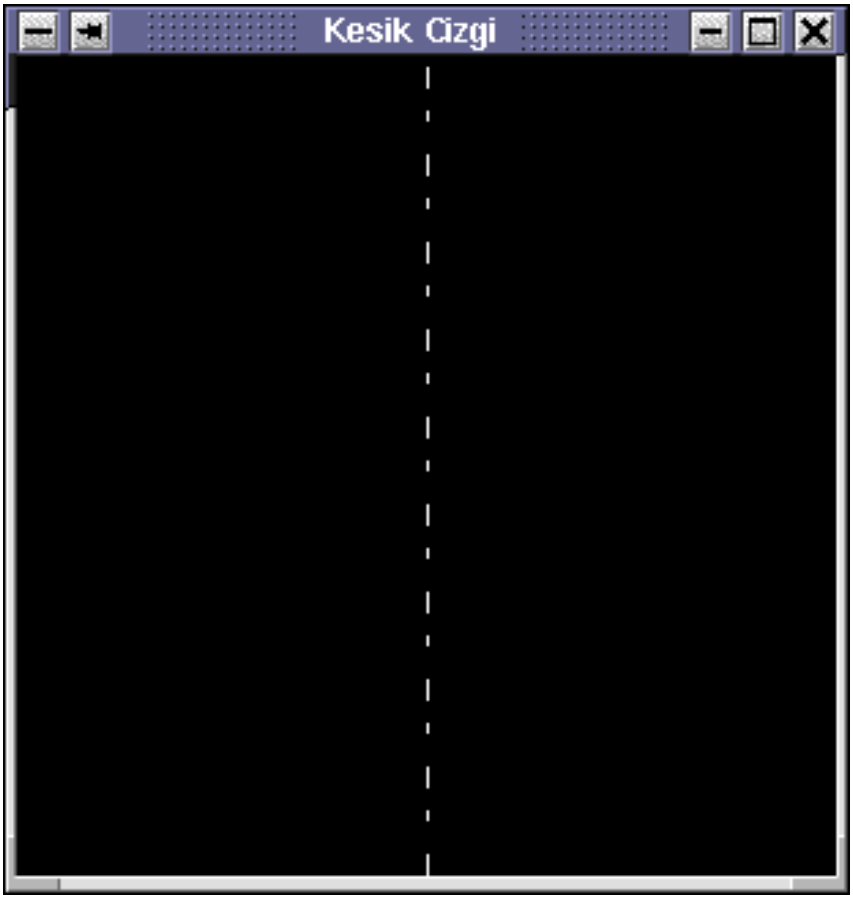
glilk.c

```
#include <GL/glut.h>

void myDisplay()
{
  glClear(GL_COLOR_BUFFER_BIT);
  glColor3f (0,0,1);
  glBegin(GL_POLYGON);
    glVertex2f (0.0, 0.0);
    glVertex2f (0.2, 0.0);
    glVertex2f (0.2, 0.5);
  glEnd();

  glFlush();
}

int main (int argc, char ** argv)
{
  glutInit (&argc,argv);
  glutCreateWindow("Iste ilk pencerem!");
  glutDisplayFunc(myDisplay);
  glutMainLoop();
  return(0);
}
```



glKesikCizgi.c

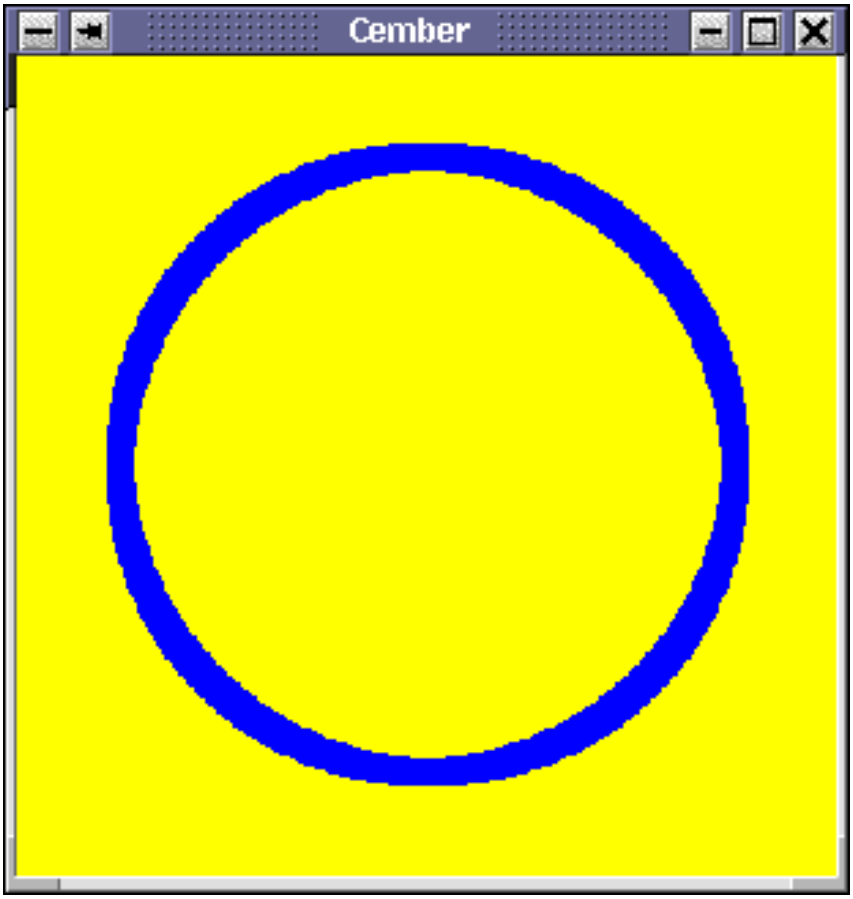
```
#include <GL/glut.h>

void display(void)
{
  glClear(GL_COLOR_BUFFER_BIT);

  glColor3f(1,1,1);// white
  glEnable(GL_LINE_STIPPLE);
  glLineStipple (2, 0x0C0F);
  glBegin(GL_LINE_STRIP);
    glVertex2f(0,-1);
    glVertex2f(0,1);
  glEnd();

glFlush();
}

int main(int argc, char **argv)
{
  glutInit(&argc, argv);
  glutCreateWindow("Kesik Cizgi");
  glutDisplayFunc(display);
  glutMainLoop();
  return(0);
}
```



glCember.c

```
#include <GL/glut.h>
#include <math.h>
#include <stdio.h>

#define RADIUS 0.75

void cember(void);

int main(int argc, char ** argv)
{
  glutInit(&argc, argv);
  glutCreateWindow("Cember");
  glutDisplayFunc(cember);
  glutMainLoop();
  return(0);
}

void cember(void)
{
  double x,y;
  int i;

  glClearColor(1.0,1.0,0.0,0.0);
  glClear(GL_COLOR_BUFFER_BIT);
  glColor3f(0,0,1);// blue

  glPointSize(10.0);
```

```
  glBegin(GL_POINTS);
    for(i = 1; i < 360; i++){
      x = RADIUS * sin(((double)i)*M_PI/180);
      y = RADIUS * cos(((double)i)*M_PI/180);
      #ifdef DEBUG
        fprintf(stderr, "(%f, %f)\n", x, y);
      #endif
      glVertex2f(x,y);
    }
  glEnd();

  glFlush();
}
```



glRenkliCizgi.c

```
#include <GL/glut.h>
#include <math.h>

void RenkliCiz(void);

#define RADIUS 0.75

int main(int argc, char **argv)
{
  glutInit(&argc, argv);
  glutCreateWindow("Renkli Cizgi");
  glClearColor (0.1, 0.1, 0.1, 0.0);
  glutDisplayFunc(RenkliCiz);
  gluOrtho2D (-5.0, 5.0, -5.0, 5.0);
```

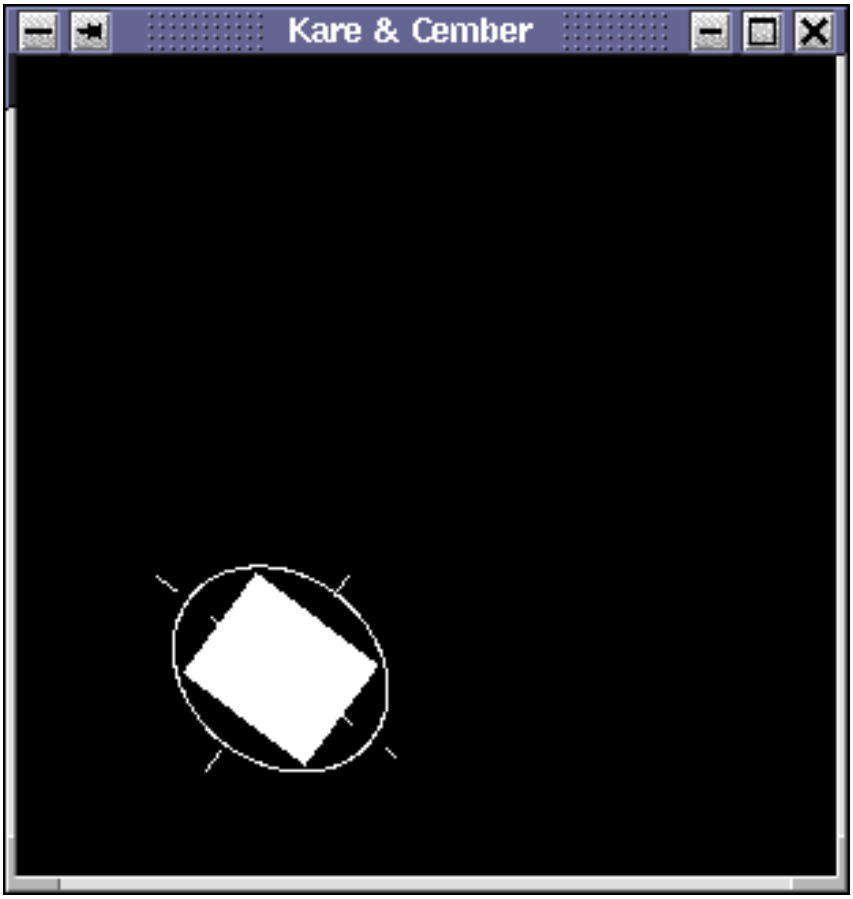
```
  glutMainLoop();
  return(0);
}

void RenkliCiz (void)
{
  int i, j, k, tur;
  double x, y;
  int teta=0;

  glClear(GL_COLOR_BUFFER_BIT);
  glPointSize(12.0);
  glBegin(GL_POINTS);
    for(i = 0; i <= 10; i++){
      for(j = 0; j <= 10; j++){
        for(k = 0; k <= 10; k++){
          glColor3f (i*0.1, j*0.1, k*0.1);
          x = teta; y = x;
          #ifdef DEBUG
            fprintf(stderr, "teta: %d, Color: %f, %f, %f; (%f, %f)\n",
                    teta, i*0.1, j*0.1, k*0.1, x,y);
          #endif
          tur = teta/360;
          x = RADIUS*(tur+1)*sin(((double)(teta-tur*360))*M_PI/180);
          y = RADIUS*(tur+1)*cos(((double)(teta-tur*360))*M_PI/180);
          glVertex2f(x,y);

          teta++;
        }
      }
    }
  glEnd();

  glFlush();
}
```

glKare.c

```
#include <GL/glut.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <math.h>
#include <stdio.h>

#define RADIUS 0.75

#define X0 -1
#define Y0 -1
#define X1 1
#define Y1 1


void cember (void);
void dondur (unsigned char ,int, int);

int teta;
int viewX0, viewY0, viewX1, viewY1;

int
main(int argc, char **argv)
{
  printf("Kullanim sekli:\n");
  printf("a: sol,          d: sag,          s: asagi,      w: yukari\n");
  printf("u: yukari cevir, n: asagi cevir, h: sola cevir, j: saga cevir\n");
  printf("b: duzlemde sola dondur,         m: duzlemde saga dondur\n");
```

```
  printf("-: goruntuyu kucult,              +: goruntuyu buyut\n");
  printf("r: goruntuyu ilk duruma getir, ESC: Cikis\n");

  teta=0;
  viewX0 = -2; viewX1 = 2; viewY0 = -2; viewY1 = 2;
  glutInit(&argc, argv);
  glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB);
  glutCreateWindow("Kare & Cember");

  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluOrtho2D(viewX0, viewX1, viewY0, viewY1);

  glutDisplayFunc(cember);
  glutKeyboardFunc(dondur);
  glutMainLoop();
  return(0);
}

void cember(void)
{
  double x, y;
  int i;

  glClearColor(0.0, 0.0, 0.0, 0.0);
  glClear(GL_COLOR_BUFFER_BIT);

  glColor3f(1, 1, 1);// blue

  glBegin(GL_POINTS);
    for(i = 1; i < 360; i++){
      x = RADIUS * sin(((double)i) * M_PI / 180);
      y = RADIUS * cos(((double)i) * M_PI / 180);
      glVertex2f(x, y);
    }
  glEnd();

  glColor3f(1, 1, 1);// white
  glEnable(GL_LINE_STIPPLE);
  glLineStipple (2, 0x0C0F);
  glBegin(GL_LINE_STRIP);
    glVertex2f(0, -1);
    glVertex2f(0, 1);
  glEnd();
  glBegin(GL_LINES);
    glVertex2f(-1, 0);
    glVertex2f (1, 0);
  glEnd();

  glColor3f(1.0, 1.0, 1.0);

  glRectf(-0.5, -0.5, 0.5, 0.5);
  glBegin(GL_LINES);
    glVertex2f(-1, 0);
    glVertex2f (1, 0);
  glEnd();
```

```
  glutSwapBuffers();
}

void dondur (unsigned char tus, int x, int y)
{
  if (tus == 'd'){
   glTranslatef (1.0, 0.0, 0.0);
  } else if (tus == 'a'){
     glTranslatef (-1.0, 0.0, 0.0);
  } else if (tus == 'w'){
     glTranslatef (0.0, 1.0, 0.0);
  } else if (tus == 's'){
     glTranslatef (0.0, -1.0, 0.0);
  } else if (tus == 'u'){
     glRotatef(15,1.0, 0.0, 0.0);
  } else if (tus == 'n'){
     glRotatef(15,-1.0, 0.0, 0.0);
  } else if (tus == 'h'){
     glRotatef(15, 0.0, 1.0, 0.0);
  } else if (tus == 'j'){
     glRotatef(15, 0.0, -1.0, 0.0);
  } else if (tus == 'b'){
     glRotatef(15, 0.0, 0.0, 1.0);
  } else if(tus == 'm'){
     glRotatef(15, 0.0, 0.0, -1.0);
  } else if (tus == '+'){
     glScalef(1.5, 1.5, 1.5);
  } else if (tus == '-'){
     glScalef(0.5, 0.5, 0.5);
  } else if (tus == 'r'){
     glLoadIdentity();
  } else if (tus == 27){
     exit(0);
  }
  cember();
}
```


Kare & Cember

A. Örnek Programlar

Bu bölümde benim yazmadığım ama farklı kaynaklardan elime geçen OpenGL ile yapılabileceklere örnek olacak programlar bulunmaktadır.

Aşağıdaki programların her birini derlemek için şu komut kullanılabilir:

```
# gcc DOSYAADI.c -o DOSYAADI -lGL -lGLU -lglut -lm
```

A.1. cube.c

```
/* Kullanımı: YUKARI ve AŞAĞI OK tuşları ve X, Y, Z tuşları */

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

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define WIDTH  480
#define HEIGHT 480

#define RED   0
#define GREEN 0
#define BLUE  0
#define ALPHA 1

#define KEY_ESC 27
#define KEY_UP 101
#define KEY_DOWN 103
#define KEY_X 120
#define KEY_Y 121
#define KEY_Z 122

GLuint face;
GLuint cube;

#define DELTA 5
int x = 0;
int rotateX = 0;
int y = 0;
int rotateY = 0;
int z = 0;
int rotateZ = 0;
int speed = 0;

void init_scene();
void render_scene();
GLvoid initGL();
GLvoid window_display();
GLvoid window_reshape(GLsizei width, GLsizei height);
GLvoid window_idle();
GLvoid window_key(unsigned char key, int x, int y);
GLvoid window_special_key(int key, int x, int y);
```

```
int main(int argc, char **argv)
{
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH);

  glutInitWindowSize(WIDTH, HEIGHT);
  glutInitWindowPosition(0, 0);
  glutCreateWindow("Rotating Cube");

  initGL();
  init_scene();

  glutDisplayFunc(&window_display);
  glutReshapeFunc(&window_reshape);
  glutIdleFunc(&window_idle);
  glutKeyboardFunc(&window_key);
  glutSpecialFunc(&window_special_key);

  glutMainLoop();

  return 1;
}

GLvoid initGL()
{
  glClearColor(RED, GREEN, BLUE, ALPHA);
  glClearDepth(1.0);
  glDepthFunc(GL_LESS);
  glEnable(GL_DEPTH_TEST);
}

void init_scene()
{
  face = glGenLists(2);
  cube = face + 1;

  glNewList(face, GL_COMPILE);
    glBegin(GL_POLYGON);
      glVertex3f(0, 0, 0);
      glVertex3f(1, 0, 0);
      glVertex3f(1, 1, 0);
      glVertex3f(0, 1, 0);
    glEnd();
  glEndList();

  glNewList(cube, GL_COMPILE);
    glTranslatef(-0.5, -0.5, 0.5);

    glColor3f(1, 0, 0);
    glCallList(face);

    glColor3f(1, 1, 0);
    glPushMatrix();
    glTranslatef(0, 0, -1);
    glCallList(face);
    glPopMatrix();
```

```
    glColor3f(0, 1, 0);
    glPushMatrix();
    glRotatef(90, 0, 1, 0);
    glCallList(face);
    glPopMatrix();

    glColor3f(0, 1, 1);
    glPushMatrix();
    glTranslatef(1, 0, 0);
    glRotatef(90, 0, 1, 0);
    glCallList(face);
    glPopMatrix();

    glColor3f(0, 0, 1);
    glPushMatrix();
    glRotatef(-90, 1, 0, 0);
    glCallList(face);
    glPopMatrix();

    glColor3f(1, 0, 1);
    glPushMatrix();
    glTranslatef(0, 1, 0);
    glRotatef(-90, 1, 0, 0);
    glCallList(face);
    glPopMatrix();

  glEndList();
}

GLvoid window_display()
{
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glLoadIdentity();
  gluLookAt(0, 0, 5, 0, 0, 0, 0, 1, 0);
  render_scene();
  glutSwapBuffers();
}

GLvoid window_reshape(GLsizei width, GLsizei height)
{
  if (height == 0)
    height = 1;

  glViewport(0, 0, width, height);

  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluPerspective(45, (GLdouble)width/(GLdouble)height, 1, 10);
  glMatrixMode(GL_MODELVIEW);
}

GLvoid window_key(unsigned char key, int x, int y)
{
  switch (key) {
  case KEY_ESC:
    exit(1);
    break;
```

```
  case KEY_X:
    rotateX = !rotateX;
    glutPostRedisplay();
    break;
  case KEY_Y:
    rotateY = !rotateY;
    glutPostRedisplay();
    break;
  case KEY_Z:
    rotateZ = !rotateZ;
    glutPostRedisplay();
    break;
  default:
    printf ("Pressing %d doesn't do anything.\n", key);
    break;
  }
}

GLvoid window_special_key(int key, int x, int y)
{
  switch (key) {
  case KEY_UP:
    speed = (speed + DELTA + 360) % 360;
    glutPostRedisplay();
    break;

  case KEY_DOWN:
    speed = (speed - DELTA + 360) % 360;
    glutPostRedisplay();
    break;

  default:
    printf ("Pressing %d doesn't do anything.\n", key);
    break;
  }
}

GLvoid window_idle()
{
  if (rotateX) x = (x + speed + 360) % 360;
  if (rotateY) y = (y + speed + 360) % 360;
  if (rotateZ) z = (z + speed + 360) % 360;
  if (speed > 0 && (rotateX || rotateY || rotateZ))
    glutPostRedisplay();
}

void render_scene()
{
  glRotatef(x, 1, 0, 0);
  glRotatef(y, 0, 1, 0);
  glRotatef(z, 0, 0, 1);
  glCallList(cube);
}
```

A.2. glorgMolehill.c

```
#include <GL/glut.h>

GLfloat mat_red_diffuse[] = { 0.7, 0.0, 0.1, 1.0 };
GLfloat mat_green_diffuse[] = { 0.0, 0.7, 0.1, 1.0 };
GLfloat mat_blue_diffuse[] = { 0.0, 0.1, 0.7, 1.0 };
GLfloat mat_yellow_diffuse[] = { 0.7, 0.8, 0.1, 1.0 };
GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat mat_shininess[] = { 100.0 };
GLfloat knots[8] = { 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0 };
GLfloat pts1[4][4][3], pts2[4][4][3];
GLfloat pts3[4][4][3], pts4[4][4][3];
GLUnurbsObj *nurb;
int u, v;

static void
display(void)
{
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glCallList(1);
  glFlush();
}

int
main(int argc, char **argv)
{
  glutInit(&argc, argv);
  glutCreateWindow("molehill");
  glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
  glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);
  glEnable(GL_LIGHTING);
  glEnable(GL_LIGHT0);
  glEnable(GL_DEPTH_TEST);
  glEnable(GL_AUTO_NORMAL);
  glEnable(GL_NORMALIZE);
  nurb = gluNewNurbsRenderer();
  gluNurbsProperty(nurb, GLU_SAMPLING_TOLERANCE, 25.0);
  gluNurbsProperty(nurb, GLU_DISPLAY_MODE, GLU_FILL);

  /* Build control points for NURBS mole hills. */
  for(u=0; u<4; u++) {
    for(v=0; v<4; v++) {
      /* Red. */
      pts1[u][v][0] = 2.0*((GLfloat)u);
      pts1[u][v][1] = 2.0*((GLfloat)v);
      if((u==1 || u == 2) && (v == 1 || v == 2))
        /* Stretch up middle. */
        pts1[u][v][2] = 6.0;
      else
        pts1[u][v][2] = 0.0;

      /* Green. */
      pts2[u][v][0] = 2.0*((GLfloat)u - 3.0);
      pts2[u][v][1] = 2.0*((GLfloat)v - 3.0);
      if((u==1 || u == 2) && (v == 1 || v == 2))
        if(u == 1 && v == 1)
          /* Pull hard on single middle square. */
          pts2[u][v][2] = 15.0;
```

```
        else
          /* Push down on other middle squares. */
          pts2[u][v][2] = -2.0;
      else
        pts2[u][v][2] = 0.0;

      /* Blue. */
      pts3[u][v][0] = 2.0*((GLfloat)u - 3.0);
      pts3[u][v][1] = 2.0*((GLfloat)v);
      if((u==1 || u == 2) && (v == 1 || v == 2))
        if(u == 1 && v == 2)
          /* Pull up on single middple square. */
          pts3[u][v][2] = 11.0;
        else
          /* Pull up slightly on other middle squares. */
          pts3[u][v][2] = 2.0;
      else
        pts3[u][v][2] = 0.0;

      /* Yellow. */
      pts4[u][v][0] = 2.0*((GLfloat)u);
      pts4[u][v][1] = 2.0*((GLfloat)v - 3.0);
      if((u==1 || u == 2 || u == 3) && (v == 1 || v == 2))
        if(v == 1)
          /* Push down front middle and right squares. */
          pts4[u][v][2] = -2.0;
        else
          /* Pull up back middle and right squares. */
          pts4[u][v][2] = 5.0;
      else
        pts4[u][v][2] = 0.0;
    }
  }
  /* Stretch up red's far right corner. */
  pts1[3][3][2] = 6;
  /* Pull down green's near left corner a little. */
  pts2[0][0][2] = -2;
  /* Turn up meeting of four corners. */
  pts1[0][0][2] = 1;
  pts2[3][3][2] = 1;
  pts3[3][0][2] = 1;
  pts4[0][3][2] = 1;

  glMatrixMode(GL_PROJECTION);
  gluPerspective(55.0, 1.0, 2.0, 24.0);
  glMatrixMode(GL_MODELVIEW);
  glTranslatef(0.0, 0.0, -15.0);
  glRotatef(330.0, 1.0, 0.0, 0.0);

  glNewList(1, GL_COMPILE);
    /* Render red hill. */
    glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_red_diffuse);
    gluBeginSurface(nurb);
      gluNurbsSurface(nurb, 8, knots, 8, knots,
        4 * 3, 3, &pts1[0][0][0],
        4, 4, GL_MAP2_VERTEX_3);
    gluEndSurface(nurb);
```

```
    /* Render green hill. */
    glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_green_diffuse);
    gluBeginSurface(nurb);
      gluNurbsSurface(nurb, 8, knots, 8, knots,
        4 * 3, 3, &pts2[0][0][0],
        4, 4, GL_MAP2_VERTEX_3);
    gluEndSurface(nurb);

    /* Render blue hill. */
    glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_blue_diffuse);
    gluBeginSurface(nurb);
      gluNurbsSurface(nurb, 8, knots, 8, knots,
        4 * 3, 3, &pts3[0][0][0],
        4, 4, GL_MAP2_VERTEX_3);
    gluEndSurface(nurb);

    /* Render yellow hill. */
    glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_yellow_diffuse);
    gluBeginSurface(nurb);
      gluNurbsSurface(nurb, 8, knots, 8, knots,
        4 * 3, 3, &pts4[0][0][0],
        4, 4, GL_MAP2_VERTEX_3);
    gluEndSurface(nurb);
  glEndList();

  glutDisplayFunc(display);
  glutMainLoop();
  return 0;             /* ANSI C requires main to return int. */
}
```

A.3. lesson4.c

```
//
// This code was created by Jeff Molofee '99
// (ported to Linux/GLUT by Richard Campbell '99)
//
// If you've found this code useful, please let me know.
//
// Visit me at www.demonews.com/hosted/nehe
// (email Richard Campbell at ulmont@bellsouth.net)
//
#include <GL/glut.h>    // Header File For The GLUT Library
#include <GL/gl.h>   // Header File For The OpenGL32 Library
#include <GL/glu.h>  // Header File For The GLu32 Library
#include <unistd.h>     // Header File For sleeping.

/* ASCII code for the escape key. */
#define ESCAPE 27

/* The number of our GLUT window */
int window;

/* rotation angle for the triangle. */
float rtri = 0.0f;
```

```
/* rotation angle for the quadrilateral. */
float rquad = 0.0f;

/* A general OpenGL initialization function.
   Sets all of the initial parameters. */
void InitGL(int Width, int Height)       // We call this right after our OpenGL
                                         // window is created.
{
  glClearColor(0.0f, 0.0f, 0.0f, 0.0f);  // This Will Clear The Background
                                         // Color To Black
  glClearDepth(1.0);                     // Enables Clearing Of The Depth Buffer
  glDepthFunc(GL_LESS);                  // The Type Of Depth Test To Do
  glEnable(GL_DEPTH_TEST);               // Enables Depth Testing
  glShadeModel(GL_SMOOTH);               // Enables Smooth Color Shading

  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();                      // Reset The Projection Matrix

  gluPerspective(45.0f,(GLfloat)Width/(GLfloat)Height,0.1f,100.0f);
                                  // Calculate The Aspect Ratio Of The Window

  glMatrixMode(GL_MODELVIEW);
}

/* The function called when our window is resized
   (which shouldn't happen, because we're fullscreen) */
void ReSizeGLScene(int Width, int Height)
{
  if (Height==0)         // Prevent A Divide By Zero If The Window Is Too Small
    Height=1;

  glViewport(0, 0, Width, Height);  // Reset The Current Viewport And
                                    // Perspective Transformation
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();

  gluPerspective(45.0f,(GLfloat)Width/(GLfloat)Height,0.1f,100.0f);
  glMatrixMode(GL_MODELVIEW);
}

/* The main drawing function. */
void DrawGLScene()
{
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);   // Clear The Screen And
                                                        // The Depth Buffer
  glLoadIdentity();                 // Reset The View

  glTranslatef(-1.5f,0.0f,-6.0f); // Move Left 1.5 Units And Into The Screen 6.0

  glRotatef(rtri,0.0f,1.0f,0.0f);   // Rotate The Triangle On The Y axis
  // draw a triangle (in smooth coloring mode)
  glBegin(GL_POLYGON);              // start drawing a polygon
  glColor3f(1.0f,0.0f,0.0f);        // Set The Color To Red
  glVertex3f( 0.0f, 1.0f, 0.0f);    // Top
  glColor3f(0.0f,1.0f,0.0f);        // Set The Color To Green
  glVertex3f( 1.0f,-1.0f, 0.0f);    // Bottom Right
  glColor3f(0.0f,0.0f,1.0f);        // Set The Color To Blue
```

```
  glVertex3f(-1.0f,-1.0f, 0.0f);    // Bottom Left
  glEnd();                          // we're done with the polygon (smooth
                                    // color interpolation)
  glLoadIdentity();                 // make sure we're no longer rotated.
  glTranslatef(1.5f,0.0f,-6.0f);    // Move Right 3 Units, and back into the
                                    // screen 6.0
  glRotatef(rquad,1.0f,0.0f,0.0f); // Rotate The Quad On The X axis
  // draw a square (quadrilateral)
  glColor3f(0.5f,0.5f,1.0f);        // set color to a blue shade.
  glBegin(GL_QUADS);                // start drawing a polygon (4 sided)
  glVertex3f(-1.0f, 1.0f, 0.0f);    // Top Left
  glVertex3f( 1.0f, 1.0f, 0.0f);    // Top Right
  glVertex3f( 1.0f,-1.0f, 0.0f);    // Bottom Right
  glVertex3f(-1.0f,-1.0f, 0.0f);    // Bottom Left
  glEnd();                          // done with the polygon

  rtri+=15.0f;              // Increase The Rotation Variable For The Triangle
  rquad-=15.0f;             // Decrease The Rotation Variable For The Quad

  // swap the buffers to display, since double buffering is used.
  glutSwapBuffers();
}

/* The function called whenever a key is pressed. */
void keyPressed(unsigned char key, int x, int y)
{
    /* sleep to avoid thrashing this procedure */
    usleep(100);

    /* If escape is pressed, kill everything. */
    if (key == ESCAPE)
    {
        /* shut down our window */
        glutDestroyWindow(window);

        /* exit the program...normal termination. */
        exit(0);
    }
}

int main(int argc, char **argv)
{
  /* Initialize GLUT state - glut will take any command line arguments
     that pertain to it or X Windows - look at its documentation at
     http://reality.sgi.com/mjk/spec3/spec3.html */
  glutInit(&argc, argv);

  /* Select type of Display mode:
     Double buffer
     RGBA color
     Alpha components supported
     Depth buffer */
  glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE | GLUT_ALPHA | GLUT_DEPTH);

  /* get a 640 x 480 window */
  glutInitWindowSize(640, 480);
```

```
  /* the window starts at the upper left corner of the screen */
  glutInitWindowPosition(0, 0);

  /* Open a window */
  window = glutCreateWindow("Jeff Molofee's GL Code Tutorial ... NeHe '99");

  /* Register the function to do all our OpenGL drawing. */
  glutDisplayFunc(&DrawGLScene);

  /* Go fullscreen.  This is as soon as possible. */
  glutFullScreen();

  /* Even if there are no events, redraw our gl scene. */
  glutIdleFunc(&DrawGLScene);

  /* Register the function called when our window is resized. */
  glutReshapeFunc(&ReSizeGLScene);

  /* Register the function called when the keyboard is pressed. */
  glutKeyboardFunc(&keyPressed);

  /* Initialize our window. */
  InitGL(640, 480);

  /* Start Event Processing Engine */
  glutMainLoop();

  return 1;
}
```

B. Faydalanılan Kaynaklar

[1]
OpenGL: A Primer -- **Edward ANGEL** -- ISBN: 0201741865 --

[2]
http://www.opengl.org/developers/about/overview.html --

Bilgi alınabilecek diğer adresler

http://www.google.com
: Arama motoru

http://www.whatis.com
: Bilisim terimleri sözlüğü

http://www.opengl.org
: Kitaplığın resmi web sayfası

http://www.mesa3d.org
: OpenGL destekleyen özgur yazılım uygulamasının resmi sayfası.

http://www.cevis.uni-«bremen.de/~uwe/opengl/opengl.html
: OpenGL işlevlerinin yardım sayfaları

http://www.eecs.tulane.edu/www/Terry/OpenGL/Introduction.html
OpenGL programlamaya giriş belgesi

Notlar

a) Belge içinde dipnotlar ve dış bağlantılar varsa, bunlarla ilgili bilgiler bulundukları sayfanın sonunda dipnot olarak verilmeyip, hepsi toplu olarak burada listelenmiş olacaktır.

b) Konsol görüntüsünü temsil eden sarı zeminli alanlarda metin genişliğine sığmayan satırların sığmayan kısmı ¬ karakteri kullanılarak bir alt satıra indirilmiştir. Sarı zeminli alanlarda ¬ karakteri ile başlayan satırlar bir önceki satırın devamı olarak ele alınmalıdır.

Bu dosya (opengl–giris.pdf), belgenin XML biçiminin TEXLive ve belgeler-xsl paketlerindeki araçlar kullanılarak PDF biçimine dönüştürülmesiyle elde edilmiştir.

9 Şubat 2007