OPENGL NEDİR?

OpenGL bilgisayar grafiği çizmek için bir API'dir.

Temel geometrik şekiller ve görüntüler ile yüksek kalitede renkli görüntüler oluşturur.

3 boyutlu grafikler ile etkileşimi uygulamalar yaratır.

İşletim sisteminden ve pencere sisteminden bağımsızdır.

NASIL ÇALIŞIR?

OpenGL'in nasıl çalıştığını anlamak için onun bir çizimi nasıl

oluşturduğuna bakmak gerekli.OpenGL de bir şey çizmek için ***geometrik ilkelleri*** (geometric primitives -noktalar, çizgiler ve çokgenler) ve ***görüntü ilkelleri*** (resimler ,bitmaplar) kullanılır. Daha sonra OpenGL'in durum mekanizması olarak adlandırılan mekanizmaya başvurularak çizilecek objenin renk,madde, ışık kaynağı ve bunun gibi özellikleri ayarlanır ve yönetilir.

OpenGL Temel Kütüphaneleri ve İşlevleri

- **GL :** OpenGL'in platform bağımsız "çekirdek" kütüphanesidir.
- **GLU (OpenGL Utility Library):** Değişik grafik fonksiyonlarını içeren yardımcı kütüphane. OpenGL'in bir parçasıdır.Eğriler, yüzeyler, yüzey döşeme görüntüleri,vb içerir.
- **GLUT(OpenGL Utility Toolkit)/AUX :** İşletim sisteminin pencereleme sistemi tarafından sağlanan pencerelerin OpenGL programları tarafından yönetimini sağlayan "utility toolkit" kütüphaneleridir.Daha yalın bir pencere sistemi bütünleştirmesi sunar.Resmi anlamda OpenGL'in parçası değildir
- **AGL, GLX, WGL:** Xwindow ve MS Windows ortamları için

OpenGL ile pencere sistemleri arasındaki katmandır.

GL Kütüphanesi Organizasyonu

- **X Windows Altında**

X Window grafik sisteminde uygulama programı temel ve yardımcı OpenGL kütüphaneleri yardımıyla işletim sisteminin grafik kütüphaneleri olan Xlib ve Xtk ile etkileşimde bulunur.

- **MS Windows Altında**

Windows ortamında durum biraz farklıdır. Win32 sisteminde uygulama programı OpenGL'in temel ve yardımcı kütüphanelerinin yanısıra direk olarak işletim sisteminin kaynaklarından faydalanabilmektedir.

OpenGL Grafik API'sinin Sağladığı Özellikler

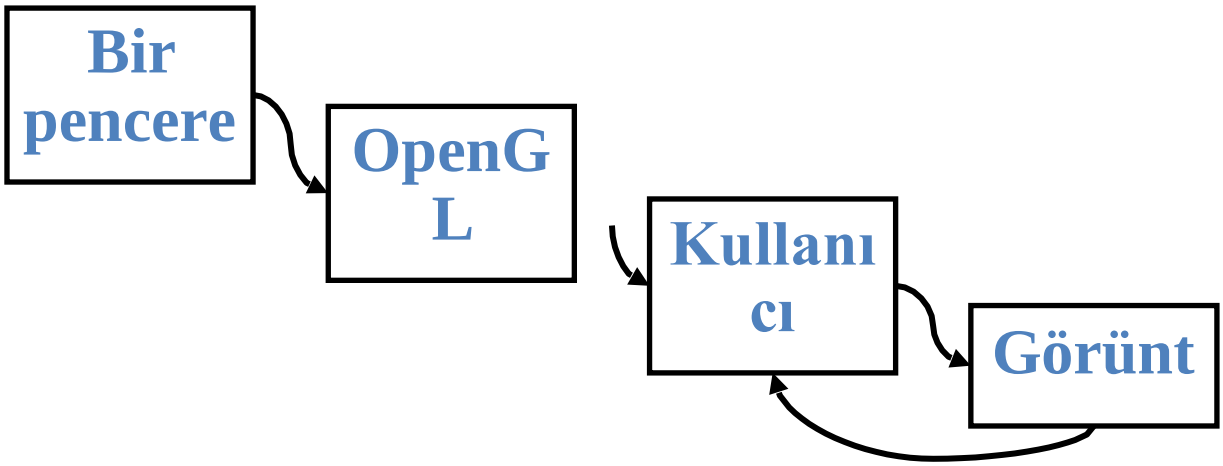
- **Primitifler**
  - nokta, çizgi, çokgen, bitmap, veya görüntü.
- **Transformasyon işlemleri**
  - Döndürme, yerdeğiştirme, 3D koordinat uzayında bakış açısı.
- **Renk modu**
  - RGB, RGBA, Color index.
- Maddelerin ışıklandırılması ve gölgelendirilmesi
- **Bufferlama**
  - Double buffering, Z-buffering, Accumulation buffer.
- **Texture mapping**

OpenGL Boru Hatlari

OpenGL teknolojisinin iki adet çalışma biçimi vardır. Bunlara "OpenGL pipeline" denir ,yani OpenGL boru hattı tabiri kullanılır. Bunlar :

- Piksel-tabanlı, görüntü ilkelleri(primitive) boru hattı.
- Geometrik, vertex-tabanlı görüntü ilkelleri.
- Texture- her iki boru hattını birleştirerek kullanır.Çünkü basit görüntü ve bitmaplar pixel tabanlı görüntülerdir ve geometrik modellere uygulanırlar.

OpenGL Programının Genel Yapısı



OpenGL Programı Tasarım Aşamaları

Genel olarak herhangi bir OpenGL programı yazılırken aşağıdaki adımlar takib edilir :

1. **1-GL ve GLUT ayarlanır :** Pencere açma, görüntüleme modu, ……
2. **2-OpenGL'i başlangıç duruma getirme :** Arka renk, ışık, bakma
3. pozisyonları, ……
4. **3-Callback fonksiyonlarını kayda geçirme :** Renderleme fonksiyonu, etkileşim (klayve, fare)

4-Olay işleme döngüsü: glutMainLoop()fonksiyonu yazılır.

OpenGL'İ Başlangıç Durumuna Getirme

```
void myinit(void)
{
    glClearColor(1.0, 1.0, 1.0, 1.0);          // arka renk
    glColor3f(1.0, 0.0, 0.0);                  // çizgi rengi
    glMatrixMode(GL_PROJECTION);               // görüntüleme modeli
    glLoadIdentity();
```

```
    gluOrtho2D(0.0, 500.0, 0.0, 500.0);

    glMatrixMode(GL_MODELVIEW);

}
```

Callback Fonksiyonları

Bir olay olduğunda çağırılacak kod rutinleridir. OpenGL tarafından sağlanan standard callback fonksiyonları vardır ancak programcı kendi istediği callback fonksiyonlarını da yazabilir.

GLUT-olay işlemeyi sürdürmek için callback mekanizmasını kullanır**.**

Olay Süreci Döngüsü

OpenGL uygulamalarında uygulamanın olayları algıladığı bloktur. Uygulama callback fonksiyon çağrılarını buradan yönetir.Sistemde herhangi bir olay veya çağırı oluştuğunda bu olaylar *"olay süreci döngüsü"* 'ne iletilirler. Bu mekanizmada uygun callback fonksiyonunu meydana gelen olayla ilişkilendirir.

```
void main (int argc, char **argv)

{

    ……

    glutMainLoop();

}
```

Vertex Kavramı

Vertex :nokta, uzayda bir yer anlamına gelir. OpenGL de oluşturulan bütün görüntüler *vertex*'ler vasıtasıyla oluşturulur. Vertexler glVertex*() komutu ile tanımlanır.Burada * işaretinin yerine 2f yada 3f olabilir.

```
örnek:        glBegin(GL_LİNES);

              glVertex2f(0.0,0.0);

              glVertex2f(0.1,0.1);

            glEnd();
```

Bu örnekte glBegin() fonksiyonu aşağıda verilen primitiflerden herhangisini alabilir. glBegin()'in aldığı primitif, vertexler ile belirlenen noktaları birleştirmede kullanılır.

Geometrik Primitifler

Geometrik primitifler bir çift glBegin() ve glEnd() fonksiyonları arasında tanımlanır. Bu iki komut arasında geometrik primitifin koordinatlarını belirleyen bir veya birden fazla glVertex*() komutu yer alır.

```
glBegin( tip );
    glVertex*(…);
    ……
    glVertex*(…);
glEnd();
```

Tip parametresi, vertix'lerin nasıl birleştiğini belirtir.

OpenGL Geometrik Primitiflerinin İşlevleri

- GL_POINTS: her vertex ayrı bir nokta olarak ele alır
- GL_LİNES: her vertex çiftini bağımsız çizgi olarak ele alır
- GL_LINE_STRIP : ilk vertex ile son vertex'i çizgilerle birleştirir
- GL_LINE_LOOP : ilk ve son vertex aralarını çizgilerle birleştirir ve en son olarak baş vertexe döner.
- GL_POLYGON : vertex koordinatlarını kullanarak tek bir çokgen çizer
- GL_QUADS : her dört vertex bağımsız bir dörtgen olarak ele alınır
- GL_TRIANGLES : her üç vertex bağımsız bir üçgen olarak ele alınır
- GL_TRIANGLE_FAN : bir birine bağlı üçgenler kümesi oluşturur.

Diğer OpenGL Özellikleri

- glClearColor(x, y, z, a);

- Art alanı boyama fonksiyonudur
- 4. argüman saydamlığı belirler; 1.0 tamamen saydam
- Durum değişkenini set eder

□ glPointSize(x);

- Nokta büyüklüğünü "X" piksele ayarlar
- Not: bu cihaz bağımsız bir özellik değildir. Yani çihazlara göre değişir
- Durum değişkenini set eder

□ glLinewidth (x);

-çizgi genişliğini x değeriyle belirler.

OpenGL Renk Modeli

□ OpenGL'da 2 renk modeli vardır:

- RGB (Gerçek Renk): RGB renk Modeli'nde her pikselin "Red, Green, Blue",yani "Kırmızı,Yeşil,Mavi" bileşenleri mevcüttur .RGB modunda, bir pikselin rengi diğerlerinden bağımsızdır.Yani her piksel için ayrı ayrı renk özellikleri atanabilir.
- Indexed Color (Renk Haritası):Color-index modu RGB moduna nazaran daha az hafıza kapladığı için önemlidir.Bunun nedeni de Color-map **olarak** tanımlanan renk başvuru tablosunu kullanmasıdır. Color-index modunda aynı index'e sahip her piksel "color-map" ta aynı yuvayı kullanır(pointer kavramı gibi düşünülebilir).
- Pencerenin renk modeli pencereleme sistemi tarafından sağlanır.

RGB Renk Komutları

glColor*(…) fonksiyonu; vertex'lerin rengini belirler.Bir kere çağırılan glColor*() fonksiyonu, bir sonraki çağırma gelene kadar aynı rengi korur.

glClearColor(r, g, b, a) : pencerenin art alanının rengini belirler.

glutInitDisplayMode(mod) :Görüntülem renk modunu belirler.RGBA isteniyorsa mod olarak GLUT_RGBA , İndexed color isteniyorsa mod olarak GLUT_INDEX parametresi alınır.

OpenGL Durum Makınesi (State Machine)

OpenGL bir durum makinesi gibi çalışır. Durum, uygulamalar tarafından değiştirilebilen bir takım değişkenlere sahiptir.Örnegin glClearColor*() fonksiyonunu kullanarak renk bufferini mavi renkle silersek durum makinesinin uygun değişkeni mavi değeri alır ve bir sonraki komut gelene kadar durumunu korur. *Geçerli görüntüleme konumu, çizgi genişliği, madde özellikleri, temizleme rengi* gibi özellikler durum makinesinin diğer ögeleridir.

OpenGL'da Durum Yönetimi

- Vertex özelliklerini belirlemek için aşağıdaki

fonksiyonlar kullanılır.

  - glPointSize(...) = nokta boyutu
  - glLineWidth(…) =çizgi genişliği
  - glColor*(…) =renk
  - glNormal*(…) =normal
  - glTexCoord*(…) =texturing

- Durum Fonksiyonlarını yönetmek için ise glEnable(...) ve glDisable(...) komutları kullanılır glEnable() fonksiyonu durum fonksiyonunu aktif yaparken glDisable() fonksiyonu durum fonksiyonunu pasif yapar.

GLUT Programlama

□ GLUT (OpenGL Utility Toolkit), OpenGL Programming for the X Window System and The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics'in yazarı ve Silicon Graphics çalışanı Mark J. Kilgard tarafından OpenGL programlarının

geliştirilmesi amacıyla yazılmış bir kütüphanedir. Nate Robins tarafından Win32 platformuna da aktarılmıştır.

- GLUT kütüphanesinin iki temel amacı vardır.
- Platformlar arası kod taşınabilirliğini artırmak.
- OpenGL programlama öğrenimini kolaylaştırmak.
- GLUT kullanılarak geliştirilen programlarda işletim sistemine özgü karmaşık pencere API' leriyle uğraşmaya gerek duyulmaz. Temel olarak pencere tanımlama, pencere kontrolü, klavye, fare girişi ve izlenmesi fonksiyonlarını barındırır. Aynı zamanda birçok ilkel geometrik model için fonksiyonlar içerir.
- Tüm GLUT fonksiyonları glut ön adıyla başlar.

GLUT, açık kaynaklı bir kütüphane değildir.

GLUT Fonksiyonlarının Sınıflandırılması

GLUT fonksiyonları, işlevselliklerine göre, bir kaç alt API'ye sınıflandırılabilirler:

- Başlatım
- Başlangıç Olay İşlemesi
- Pencere İşletimi
- Örtükatman (overlay) İşletimi
- Menü İşletimi
- Callback Fonksiyonlarını Kayda Geçirme
- Renk İndeksi, Renk Haritası İşletimi
- Duruma Geridönüş
- Font Görüntüleştirimi
- Geometric Biçim Görüntüleştirimi

GLUT Başlatma Fonksiyonları

- 1. Standard GLUT başlatma;uygulamanın komut satırından bilgi almasını ve sistemi başlatmasını sağlar.
  - glutInit (int argc, char ** argv)
- 2. Görüntüleme modeli; pencere için çizim özelliklerini belirler.
  - glutInitDisplayMode (unsigned int mode)
- 3.Pencere boyutu ve yeri
  - glutInitWindowSize (int width, int height) ;piksel olarak pencere boyutu
  - glutInitWindowPosition(int x, int y) ; ekranın sol üst köşesinden uzaklığı
- 4.Pencereyi oluştur ; “simple” adındaki pencerenin yaratılması
  - glutCreateWindow (char *name);

glutInitDisplayMode

glutInitDisplayMode(unsigned int mode); GLUT gösterim modu olup bit maskelerinin bit türüdür.Olabilicek bit maske değerleri aşağıda verilmektedir :

- GLUT_RGBA : Bir RGBA mode pencere seçer. Bu, ne GLUT_RGBAne de
- GLUT_INDEX belirtilmedikçe, benimsenen değer olarak kalır
- GLUT_RGB :GLUT_RGBA ile aynı.
- GLUT_INDEX : Renk indeks pencere modunu seçer.GLUT_RGBA'ya baskındır.
- GLUT_SINGLE:Tek bir tamponlu pencere seçer. Bu benimsenen yapıdır.
- GLUT_DOUBLE: Çift tamponlanmış bir pencere seçer. GLUT_SINGLE'a baskındır.
- GLUT_ACCUM :Depolama tamponlu bir pencere seçer.
- GLUT_ALPHA : Renk tamponuna(larına) alfa bileşenli bir pencere seçer.
- GLUT_DEPTH: Derinlik tamponlu bir pencere seçer.
- GLUT_STENCIL :Örüntü yedekleme tamponlu bir pencere seçer.
- GLUT_MULTISAMPLE :Multismapling destekli bir pencere seçer.
- GLUT_STEREO : Bir stereo pencere seçer.

GLUT_LUMINANCE : "luminance" renk modelli bir stereo pencere seçer**.**

GLUT Olay Callback FonksiyonlarI

GLUT genel olarak kullanılan input olaylarını yönetebilir

Örn:

- □ Tuş aksiyonu
- □ Fare aksiyonu
- □ Idle (hiç bir şey yokken çağırılır)

Pencere olayları (içerik, boyut veya hareket)

KLAVYE OLAYLARI

Klavye üzerinde bir tuş basıldığında çağrılır.

glutKeyboardFunc (keyboard_callback_func);

ÖZEL KLAVYE OLAYI

Klavye üzerinde fonksiyon tuşları veya yön tuşları basıldığında aktif olur.

glutSpecialFunc (special_callback_func);

FARE OLAYLARI

Fare butonu basıldığında/bırakıldığında aktif olur

glutPassiveMotionFunc (mouse_passivemotion_func);

FARE PASİF HAREKET OLAYI

Fare pencere içerisinde butona basılmadan gezdirildiğinde kullanılabilir

glutMouseFunc (mouse_callback_func);

void mouse_passivemotion_func (int x, int y)

- x, y: konum

FARE HAREKET OLAYI

Farenin, pencere içerisinde hareket ettirilmesi ve 1 yada daha fazla butonunun basılması olayı

glutMotionFunc (mouse_motion_func);

FARE GİRME OLAYI

Fare,pencere alanına girmesi ve butonlarından 1 veya daha fazlasının basılması olayı

glutEntryFunc (mouse_entry_func);

PENCERE OLAYLARI

Pencere boyutu değiştiğinde veya şekli değiştiğinde aktif olur. Objeyi yeniden çizer.“Aspect ratio”yu korur.

glutReshapeFunc (reshape_callback_func);

PENCERENİN GÖRÜNME OLAYI

Pencerenin görünme(visibility) özelliği değiştiğinde aktif olur.

glutVisibilityFunc (visibility_callback_func);

DİSPLAY CALLBACK

Pencerenin yeniden görüntülenmesi gerektiğinde çağrılır. Örneğin, pencere ilk defa açıldığında.Bu fonksiyon GLUT programında mutlaka bulunmalı.

glutDisplayFunc (display_callback_func);

IDLE CALLBACK

Bu fonksiyon hiçbirşey yapılmadığında çağırılır. Animasyon ve devamlı yenilemelerde çok kullanışlıdır.

glutIdleFunc (idle_callback_func);

Menu Status (State) Callback

Pop-up menüler kullanıcı tarafından kullanılıyor iken GLUT_MENU_IN_USE durum parametresi ile çağırılır; Kullanıcı pop-up menüyü kapattığında, GLUT_MENU_NOT_IN_USE durum parametresi ile çağırılır. x ve y parametreleri,menüyü kullanıma sokan buton basılma veya buton bırakılma olayının pencere koordinatları cinsinden

konumunu belirlerler. MenuState , MenuStatus'un deprecated şeklidir(x,y parametreleri yoktur).

glutMenuStatusFunc(int status, int x, int y);

glutMenuStateFunc(int status);

Menü Oluşturma

glutAddMenuEntry (const char *label, int value);

- label: menu biriminin kullanıcıya gösterilen ismi
- value: bu birim seçildiğinde ID'si , menü callback fonksiyonuna gönderilir.

glutAttachMenu (int button);

- button: menu ile hangi fare butonu ilişkilendirilecek
- GLUT_LEFT_BUTTON, GLUT_MIDDLE_BUTTON, GLUT_RIGHT_BUTTON

GÖRÜNTÜLEME ve DÖNÜŞÜMLER

3D Görüntüleme = Fotoğraf Çekme:

3 boyutlu görüntüleme fotoğraf çekmeye benzer.

Objenin konumlandırılması (modeling)
Kameranın yerleştirilmesi (viewing)
Lens seçimi (projection)
Görüş alanı (viewport)

Koordinat Sistemleri ve Dönüşümler:

- Görüntüyü oluşturma adımları:
  1. Geometriyi belirleme (dünya koordinatları)
  2. Kamerayı belirleme (kamera koordinatları)
  3. Görüntüleme biçimini belirleme (pencere koordinatları)
  4. Görüş alanına eşleme (ekran koordinatları)
- Bütün adımlarda dönüşümler kullanılabilir.
- Her dönüşüm koordinat sistemlerinde bir değişime denk gelir.

Dönüşümleri Belirleme:

- Dönüşümler iki farklı yöntem ile belirlenebilir:

Matrisleri belirleyerek
glLoadMatrix, glMultMatrix

- İşlemi belirleyerek

glRotate, glOrtho

- Programcı gerçek matrisin ne olduğunu bilmek zorunda değildir.

Matris İşlemleri:

- Matris yığınını (stack) belirleme

glMatrixMode(*GL_MODELVIEW* veya *GL_PROJECTION*)

- Diğer Matris veya Yığın İşlemleri

glLoadIdentity() glPushMatrix()

glPopMatrix()

- Viewport

Genellikle pencere genişliği ile aynıdır
Eğer viewport genişlik/yükseklik oranı projeksiyondakinden farklı ise nesneler olduklarından farklı biçimde görünebilirler (daha geniş, daha yüksek, …)
glViewport(*x, y, genişlik, yükseklik*)

İzdüşüm Dönüşümlerini Uygulamak:

- Orthografik izdüşüm için tipik kullanım örneği:

```
glMatrixMode( GL_PROJECTION );
glLoadIdentity();
glOrtho(left, right, bottom, top, zNear, zFar);
```

Dönüşümleri Görmek:

- Kamerayı konumlandırma

Kamera ayağını aç ve kamerayı yerleştir

- Bir sahnede hareket etmek için

Görüntüleme dönüşümünü değiştir ve sahneyi tekrar çiz

gluLookAt(eye_x, eye_y, eye_z,
aim_x, aim_y, aim_z,
up_x, up_y, up_z)

- Yukarı (up) vektör kameranın üstünün neresi olduğunu belirler.

Konumlandırma Dönüşümlerini Uygulamak:

- Nesneyi taşımak:

glTranslate{fd}(*x, y, z*)

- Nesneyi bir eksen etrafında döndürmek:

glRotate{fd}(*açı, x, y, z*)

  - Açı derece cinsinden girilir

- Nesneyi büyütmek/küçültmek, genişletmek/daraltmak ve yansımasını oluşturmak:

glScale{fd}(*x, y, z*)

Kamera yeri ve konumlandırma arasındaki ilişki:

- Kamerayı belirli bir doğrultuda taşıma ile görüntülenen dünyadaki tüm nesneleri tam tersi olan doğrultuda taşıma arasında bir fark yoktur.
- Görüntüleme dönüşümleri birçok konumlandırma dönüşümüne eşittir.

gluLookAt() komutunun yaptığı işi birçok glTranslate ve glRotate kombinasyonu ile yapabilirsiniz.

Animasyon:

① Çift renk tamponu kullanımı isteği yapılır

glutInitDisplayMode*(GLUT_RGB | GLUT_DOUBLE);*

② Renk tamponu temizlenir

glClear*(GL_COLOR_BUFFER_BIT);*

③ Sahne çizimi yapılır

④ Ön ve arka tamponlar yer değiştirilir

glutSwapBuffers();

- 2, 3 ve 4. adımlar tekrar edilerek animasyon sağlanır
  glutIdleFunc() callback kullanılır.

Derinlik Tamponu Kullanımı:

① Derinlik tamponu kullanımı isteği yapılır

glutInitDisplayMode(*GLUT_RGB* | *GLUT_DOUBLE* | *GLUT_DEPTH*);

② Özellik aktif hale getirilir
glEnable(*GL_DEPTH_TEST*);

③ Renk ve derinlik tamponları temizlenir
glClear(*GL_COLOR_BUFFER_BIT* | *GL_DEPTH_BUFFER_BIT*);

④ Sahne çizilir
④ Renk tamponları yer değiştirilir.

OpenGL ışığı nasıl kullanır:

- OpenGL'de ışıklandırma Phong ışıklandırma modeline dayanır. Çizim ilkelinin her noktasında, bir renk o ilkelin malzeme özelliklerinin ışık ayarları ile birlikte kullanılması ile hesaplanır.
- Bir noktanın ışık altındaki rengini bulmak için, o noktaya verilen renk haricinde 4 bileşenin daha bilinmesine gerek vardır. Bunlar:

Ambient
Diffuse
Specular
Emmission

Işığın Bileşenleri:

- Ambient: Hangi noktadan geldiği tam olarak belirlenemeyen (tüm yönlerden geliyormuş gibi görünen) çevresel ışık bileşenidir.
- Diffuse: Tek bir noktadan gelen ışık bileşenidir.
- Specular: Belirli bir noktadan gelen ve parlak yüzeyli nesnelerde yansıma yapan ışık bileşenidir.
- Emission: Işık yayan yüzeylerdeki ışık bileşenidir. Işık sanki nesnenin içinde parlıyor gibi algılanır.

Işık Kaynakları:

glLightfv(*ışık, özellik, değer*);

- *ışık* parametresi *n* farklı ışık değerinden birini alır

GL_LIGHT0 – GL_LIGHTn

glGetIntegerv(*GL_MAX_LIGHTS, &n*);

- *özellikler*

Renkler (GL_AMBIENT, GL_DIFFUSE, GL_SPECULAR)
Yer (GL_POSITION)
Tip (GL_SPOT_...)
Zayıflama (GL_..._ATTENUATION)

Işıkları Aktif Hale Getirme:

- Işıklandırmayı açma: glEnable(*GL_LIGHTING*);
- Işık kaynağının düğmesini açma: glEnable(*GL_LIGHTn*);

Işık Modeli Özellikleri:

- glLightModelfv(*özellik, değer*);
- İki yönlü ışıklandırma sağlamak için

GL_LIGHT_MODEL_TWO_SIDE,{0.2, 0.2, 0.2, 1.0}

- Evrensel bir çevresel ışık rengi vermek için

GL_LIGHT_MODEL_AMBIENT, GL_FALSE (GL_TRUE)

- Viewpoint değiştikçe ışığın etkisinin de değişmesi

GL_LIGHT_MODEL_LOCAL_VIEWER, GL_FALSE (GL_TRUE)

- Doku kaplamada daha iyi renk kontrolü için

GL_LIGHT_MODEL_COLOR_CONTROL, GL_SINGLE_COLOR

(GL_SEPARATE_SPECULAR_COLOR)

Display Listeleri

Display Listeleri - Dizayn Felsefesi

Display Listeler Oluşturma Ve Çalıştırma

Display Listelerini Kendi İndislerinde Yönetme

Giriş

Display listeleri daha sonra çalıştırılmak üzere oluşturulmuş OpenGL komutları grubudur. Display listesi çağırıldığında, liste içinde bulunan komutlar tanımlandıkları sıraya göre çalışmaya başlarlar. Çoğu OpenGL komutları display listelerinde saklanabilir.

Display listeleri kullanımı

Display listeleri , OpenGL komutlar kümesinin isimlendirme ve organize etmenin standart yoludur. Örneğin100 adet çizgi segmentini kullanarak bir daire çizmek istiyoruz. Display listesi kullanmadan bu kod aşağıdaki gibi yazılabilir.

```
drawCircle()
  GLint i;
  GLfloat cosine, sine;
   glBegin(GL_POLYGON);
     for(i=0;i<100;i++){
       cosine=cos(i*2*PI/100.0);
       sine=sin(i*2*PI/100.0);
       glVertex2f(cosine,sine);
     }
   glEnd();
}
```

Bu metod verimsizdir çünkü dairenin her renderlanmasında trigonometrik hesaplamaların yapılması gerekmektedir. Bunun yerine koordinatları bir tabloya kaydederek gerektiğinde bu koordinatları tablodan okumak daha verimlidir.

Asıl yapılması gereken şey daireyi bir kere çizip OpenGL'in o daireyi nasıl çizdiğini hatırlamasını sağlamak. Bu display listelerinin amacıdır.

```
#define MY_CIRCLE_LIST 1

buildCircle()

{

  GLint i;

  GLfloat cosine, sine;

  glNewList(MY_CIRCLE_LIST, GL_COMPILE);

    glBegin(GL_POLYGON);

      for(i=0;i<100;i++){

        cosine=cos(i*2*PI/100.0);

        sine=sin(i*2*PI/100.0);

        glVertex2f(cosine,sine);

      }

    glEnd();

  glEndList();  }
```

Daire çizme kodu **glNewList()** ve **glEndList()** ikilisi arasında olmalıdır. Bu bir display listesi tanımlamaktadır. **glNewList()** fonksiyonunun MY_CIRCLE_LIST argümanı display listesini tanımlayan tamsayı indeksidir. Display listesini daha sonralarda **glCallList()** komutu ile çağırabiliriz:

```
 glCallList(MY_CIRCLE_LIST);
```

Display listesi sadece OpenGL çağrılarından oluşur. Son örnekteki **cos()** ve **sin()** C fonksiyonları display listelerinde tutulmazlar. Bunun yerine koordinatlar ve diğer değişkenlerin (dizi içerikleri gibi) display listesinin derlendiği andaki değerleri dispalay listesinde tutulurlar. Böyle bir liste oluşturulup derlendikten sonra değerler değiştirilemez. Mevcut bir listeyi silebilir veya yeni bir liste tanımlayabiliriz ama var olan listeyi değiştiremeyiz.

Display Listeleri -Dizayn Felsefesi

OpenGL display listeleri performansın yükseltilmesi için tasarlanmışlardır. Performansın optimize edilmesi için OpenGL display listeleri dinamik bir veritabanı olarak değil, daha ziyade bir cash bellek olarak kullanılmaktadır. Başka bir deyişle , bir kere tanımlanan display listesi bir daha değişikliğe uğratılamaz. Eğer display listeleri değiştirilebilir olsaydı, veritabanında arama ve bellek yönetimi gerekeceğinden performans düşecekti. Display listeleri OpenGL rutinlerinin ağ ortamında kullanılmasında performansı arttırırlar,çünkü display listeleri serverda kalır ve ağ trafiğini düşürürler.

Lokal bilgisayarda ise display listelerinin verimi çok çok daha fazla olabilir çünkü listeler grafik donanımı ile en uyumlu şekle getirilerek saklanırlar. Örneğin en basitinden **glRotate*()** komutu eğer display listesinde saklanırsa , çok önemli ölçüde verimi arttırabilir çünkü döndürme matrisini oluşturmak için gereken hesaplamalar basit bir şey değildir (trigonometrik fonksiyon ve karekökler içerirler).

Display listelerinde sadece en son döndürme matrisinin saklanması yeterlidir. Küçük listeler oluşturulduğunda bazen listeye referans verme, standart programlama yoluyla aynı işlemi yapan koddan farklı olmayabilir. Bunun için en çok optimizasyon sağladığı bazı alanlar aşağıda verilmiştir :

Matris işlemleri: çoğu matris işlemleri OpenGL den invers(matrisin tersi) hesabı talebinde bulunurlar. Hesaplanan matris ve bu matrisin tersi display listelerinde saklanabilirler.

Işıklar, madde özellikleri ve ışıklandırma modelleri : Çok kompleks ışıklandırma şartlarına sahip bir sahne çizerken sahnedeki her obje için farklı bir madde ve madde özelliği kullanılabilir. Maddeleri belirlemek yavaştır çünkü çok karmaşık hesaplamalar gerektirmektedir. Eğer madde tanımlamalarını display listelerine koyarsak , her maddeler arası geçişte bu hesaplamalara gerek kalmayabilir çünkü sadece hesaplamaların sonuçları saklanmış olacaktır. Dolayısıyla aydınlatılmış sahnelerin renderlenmesi daha hızlı yapılmış olacaktır.

Texture'ler(yüzey kaplamaları) : texture'ları tanımlarken onları display listelerinde tanımlayarak verimi maksimize edebiliriz. Çünkü donanım texture formatı ile OpenGL texture formatı farklıdır ve görüntülemede bir dönüşümün yapılması gerekmektedir. Dönüştürülmüş texture'lar listelerde saklanarak verim arttırılabilir.

Display listelerinin dezavantajı, içeriklerinin değiştirilememesidir. Performansı optimize etmek için OpenGL display listeleri değiştirilemez ve içeriği okunamaz .

Display Listeler oluşturma

glNewList() ve glEndList() fonksiyonları display listesinin tanımını başını ve sonunu belirlerler. Daha sonra bu listeler glCallList() komutuna gerekli index değerini vererek çağırılırlar. Display listesi makeList() rutininde oluşturulur.

Display Liste Kullanımı

Display list içinde bulunan **glTranslatef()** rutini bir sonraki çizilecek objenin konumunu değiştirmektedir. Eğer bu komutu çağırmazsak display listesinin iki kere çağırılması şekli değiştirmez, yeni şekil eskisi üzerinden aynen çizilir. İmmediat modda çağırılan **drawLine()** rutini de kendinden önce gelen **glTranslatef()** çağrısından etkilenir. Eğer display listesine transformasyon komutlarını dahil ediliyorsa , bu komutların daha sonraki etkileri dikkate alınmalıdır.

Her **glNewList()** komutunu bir **glEndList()** takip etmelidir ve eğer yeni bir liste gerekiyorsa bu komuttan sonra tanımlanmalıdır. **glEndList()** komutunu listeyi başlatmadan önce çağırmak GL_INVALID_OPERATION hatasını oluşturur.

void glNewList (GLuint *list*, GLenum *mode*);

Display listenin başlangıcını belirtir. Bundan sonra gelen OpenGL rutinleri (l **glEndList()** komutundan önce olanlar) display listesinde saklanır(bazı özel OpenGL rutinleri hariç. Bu özel rutinler liste oluşturulmasında çalışan rutinlerdir.) *list* parametresi oluşturulacak listeyi belirler. *mode* parametresinin alabilieceği değerleri ise GL_COMPILE ve GL_COMPILE_AND_EXECUTE dir.

void glEndList (void) :display listin sonunu gösterir.

Display Listelerde Ne Tutulur?

Display listesini kurarken başlangıçta sadece kullanılan ifadelerin değerleri listede yer alır. Örneğin bir dizideki değerler daha sonra değişiyorsa , display-list değerleri değişmez.

Genel olarak referans olarak parametre geçenler ve değer döndüren fonksiyonlar listede tutulamazlar çünkü liste ,değişkenlerin tanımlandıkları sınırların dışarısında çağırılmış olabilir. Eğer liste oluştururken böye komutlar çağırılmış ise bu komutlar immediate modda çalıştırılırlar ve listeye dahil edilmezler. Aşağıda display listelerinde saklanamayan opengl komutları verilmiştir.

glDeleteLists() glIsEnabled()

glFeedbackBuffer() glIsList()

glFinish() glPixelStore()

glFlush() glReadPixels()

glGenLists() glRenderMode()

glGet*() glSelectBuffer()

Display Listelerini Çalıştırma

Display listesini oluşturduktan sonra **glCallList()** komutu ile listeyi çalıştırabilirsiniz. Doğal olarak display listesini döngüye sokablir,immediate mod çağrıları ile karıştırarak kullanabilirsiniz.

void glCallList (GLuint *list*);

list parametresi ile belirlenen dispalay listesini çalıştırır. Listedeki komutlar, nasıl kaydedildiyseler aynı şakilde (aynı sırada) çalıştırılırlar eğer **liste** tanımlı değil ise hiç bir şey olmuyor.

Display listesi, OpenGL durum değişkenlerini değiştiren çağrılar da içerebildiğinden liste çalışırken bu değişkenler de değişir. OpenGL durum değişkenlerinde yapılan değişiklikler , liste çalışmayı tamamladıysa bile aynen kalır. listenin çalışma

esnasında geçerli renk ve geçerli matris değişkenlerinde yapılan değişiklikler liste çalışmayı bitirdikten sonra bile etkilerini sürdürürler:

Display Listenin Çalışmayı Bitirdikten Sonra Durum Değişkenlerinin Etkileri Örneği

```
glNewList(listIndex,GL_COMPILE);
  glColor3f(1.0, 0.0, 0.0);
  glBegin(GL_POLYGON);
    glVertex2f(0.0,0.0);
    glVertex2f(1.0,0.0);
    glVertex2f(0.0,1.0);
  glEnd();
  glTranslatef(1.5,0.0,0.0);
glEndList();
```

Bazen bu değişikliklerin kalması istenir ama bazende listeyi çalıştırmadan önceki durum daha sonra yeniden geri getirilmek üzere kayd edilmek istenir. Bunun için **glPushAttrib()** komutunu bir grup durum değişkenlerini kaydetmek için ve **glPopAttrib()** komutunu kaydedilen değişkenlerin geri yüklenmesi için kullanabilirisiniz. Geçerli matrisi kaydetmek ve geri yüklemek için daha önce de söylendiği gibi **glPushMatrix()** ve **glPopMatrix()** komutları kullanılır.

Hiyerarşik Display Listeler

glNewList() ve glEndList() çifti arasında glCallList()glCallList() çağrısı yaparak liste içinden çalıştırılan listeler , başka bir deyişle *hiyerarşik display listeler* oluşturabilirsiniz. Hiyerarşik display listeler, bileşenlerden oluşan objeler kullanılırken ve özellikle bazı bileşenlerin birden fazla kez kullanılması gerektiği durumlarda çok yararlıdırar. Sonsuz özyinelemeden kaçınmak için içiçe çağırma seviyesi(bütün OpenGL uyarlamalaı için) en az 64 olarak belirlenmiştir.

Display Listselerini Kendi İndislerinde Yönetme

Şimdiye kadar display-liste indeksi olarak pozitif tamsayıları kullanıyorduk. Gerçek hayatta bu tehlikeli olabilir çünkü kazara kullanımda olan indeksi seçerek varolan display listesinin üstüne yazarak listenin kaybolmasına sebep olabilir. Kaza ile silmelerden kaçınmak için glGenLists() komutunu kullanılmayan bir indeks seçmek için ve glIsList() komutunu da verilen indeksin kullanımda olup olmadığını öğrenmek için kulanın. Bir liste veya liste aralığını glDeleteLists() komutu yardımıyla silebilirsiniz.

GLuint glGenLists(GLsizei *range*);

Daha önce kullanılmamış display-list indislerini bulur. *range* parametresi kaç adet indis istendiğini belirler. Döndürülen tam sayı, sürekli v boşta olan display listesi indisleri bloğunun başlangıç konumunu ifade eder. Döndürülen indislerin hepsi kullanılmış ve boş olarak işaretlenir , dolayısıyla bir sonraki glGenLists() çağrıları aynı indisleri döndürmezler(ta ki silinene kadar). Eğer istenilen sayıda indis mevcut değil ise sıfır döndürür.

GLboolean **glIsList**(GLuint *list*);

list kullanılmış ise TRUE ,değilse FALSE döndürür.

Aşağıdaki örnekte tek indis istenmektedir ve eğer o indis kullanılabilir ise yeni display listesinde kullanılmaktadır :

```
listIndex=glGenLists(1);if(listIndex!=0) {
  glNewList(listIndex,GL_COMPILE);
    ...
  glEndList();
}
```

Seçme ve Geribesleme Teknikleri

(Select & Feedback Technics)

Giriş

Bazı grafik uygulamalaı sadece iki ve üç boyutlu objelerin statik bir görüntüsünü çizerler. Bazı uygulamalar da kullanaıcıya ekranda objeler tanımlama, hareket ettirme, değiştirme, silme gibi özellikler de tanır. OpenGL tam olarak bu tür interaktif uygulamaları için dizayn edilmiştir. Ekrana çizlen objeler tipik olarak çok fazla döndürme, yerdeğiştirme ve perspektif transformasyonlara maruz kaldığından bizim için kullanıcının üç boyutlu sahnade tam olarak hangi objeyi seçmek istediğini belirlemek çok güçtür. Bize yardımcı olmak için OpenGL "Seçme" mekanizmasını sunar. Bu mekanzima otomatik olarak bize pencerenin belirli bölgesinde hangi objelerin çizildiğini söyler. Seçme tam olarak OpenGL'in bir işlem modudur; geribesleme ise başka bir böyle moddur. Geribesleme modunda , grafik donanımımızı ve OpenGL'i herzamanki rendering hesaplamalar yapmak için kullanırız. Yani bu modda , hesaplanan sonuçların görüntüyü ekrana çizilmesinde kullanmak yerine ,OpenGL bu sonuçları bize döndürür (veya geribesleme yapar).

Örneğin eğer 3-boyutlu bir objeyi ekrana değilde bir çiziciye göndermek istiyorsanız , parçaları geribesleme modunda çizmeniz gerekecek,daha sonrada çizim direktiflerini toplayarak bu direktifleri çizicinin anlayabileceği komutlara dönüştürmeniz gerekecektir.

Seçme ve geribesleme modlarında , rendering moddan farklı olarak çizim bilgileri framebuffer'a değilde uygulamaya geri döndürülmektedir. Böylece , OpenGL seçme veya geribesleme modunda iken ekran donmuş olarak kalır – çizim olmaz .

Seçme

OpenGL'in seçme mekanizmasını kullanmayı planlıyorsanız , sahnenizi önce framebuffer'a çizerek daha sonra seçme moduna girmeniz gerekir ve sahneyi yeniden çizmeniz gerekir. Seçme modunda iken framebuffer'in içeriği değşmez ta ki seçme modundan çıkılana kadar. Seçme modundan çıkıldığında OpenGL , görüntüleme hacmi ile kesişen primitiflerin listesini döndürür(görüntüleme hacmi geçerli modelview and projeksiyon matrisleri ve herhangi tanımlanan kesme(clipping planes) düzlemleri tarafından belirlenir.) Görüntüleme hacmi ile kesişen her primitif bir "seçme *hit*"'e sebep olur. Burada *hit*-kelimesi inglizce isabetli vuruş manasına gelmektedir. Primitifler listesi tamsayı-değerli *isimler* dizisi ve ilgili veri-*hit kayıtları* şeklinde döndürülür. Bu *isimler-isim yığınında* tutulurlar. İsim yığını ,seçme modunda iken primitif çizimleri için gereken komutlar çağırıldığında oluşturulur. Böylelikle isimler listesi döndürüldüğünde , bu listeyi kullanıcıların ekranda hangi primitifleri seçmiş olabileceğini belirlerken kullanırız.

Temel Adımlar

Seçme mekanizmasını kullanmak için aşağıdaki adımları takip etmeliyiz :

glSelectBuffer() komutu ile hit kayıtlarının döndürüleceği bir dizi tanımlanır.

GL_SELECT parametresini **glRenderMode()** fonksiyonu ile kullanarak seçme moduna girilmesi sağlanır

glInitNames() ve **glPushName()** fonksiyonları ile isim yığınları başlangıç duruma getirilir.

Seçme için istediğimiz görüntüleme hacmi belirlenir.

Primitif çizim komutları ve isimler yığınını yönetecek komutlar verilmeli. Her primitife uygun bir isim verilmeli.

Seçme modundan çıkarak geri döndürülen seçme verisini (hit kayıtları) işlemek.

Aşağıdaki paragraflar **glSelectBuffer()** ve **glRenderMode()** komutlarını anlatmaktadır. Bir sonraki bölümde isimler yığınını yönetmek için gerek fonksiyonlar anlatılacaktır.

void glSelectBuffer(GLsizei *size*, GLuint **buffer*);

Geri döndürülen seçme verisi için bir dizi tanımlar. **buffer** argümanı işaretsiz tamsayılar dizisine bir pointerdir,bilgiler bu diziye konur, **size** dizide tutulabilen maksimum değer sayısını belirler. **glSelectBuffer()** fonksiyonu seçme moduna girilmeden önce çağırılmalı.

GLint **glRenderMode**(GLenum **mode**);

Uygulamanın rendering, seçme veya geribesleme modunda olduğunu belirler. **mode** argümanı GL_RENDER (default), GL_SELECT, veya GL_FEEDBACK olabilir. Uygulama , **glRenderMode()** fonksiyonu değişik bir argüman ile çağırılana dek verilen modda kalır. Seçme moduna girmeden önce seçme dizisini belirlemek için **glSelectBuffer()** fonksiyonu çağırılmalı. Benzer olarak geribesleme moduna girmeden önce, **glFeedbackBuffer()** komutu geribesleme dizisini belirtmek için çağırılmalı.

İsimler Yığınını Oluşturmak

Bir önceki kısımda bahsi geçtiği gibi isimler yığını, bize döndürülen seçme bilgisi için bir veri tabanı oluşturur. İsimler yığınını oluşturmak için ilk önce glInitNames() fonksiyonu ile zemin hazırlanması lazım. Bu işlem basit olarak yığını sıfırlar ve sonra biz çağırdığımız çizim komutlarına uygun olarak tamsayı isimlerimizi bu yığına ekleriz. Tahmin edebileceğiniz gibi yığını yönetme komutları bize yığına isim ekleme (glPushName()),yığından isim çıkarma (glPopName()) ve yığının en üstündeki elemanını bir başka bir eleman ile değiştirme (glLoadName()) gibi işlemleri yapabilmemizi sağlar.

```
glInitNames();
glPushName(-1);
glPushMatrix(); /* geçerli transformasyon durumunu kaydet */
   /* istediğiniz görüntüleme hacmini burada oluşturun */
   glLoadName(1);
   drawSomeObject();
   glLoadName(2);
   drawAnotherObject();
   glLoadName(3);
   drawYetAnotherObject();
   drawJustOneMoreObject();
```

glPopMatrix (); /* bir önceki transformasyon durumunu geri yükle*/

Bu örnekte ilk iki çizilecek objenin kendi isimleri vardır , üçüncü ve dördüncü objeler ise ortak bir isim kullanmaktadırlar. Bu ayarlamada eğer 3. ve 4. objelerden biri veya her ikisi seçme hit'e sebep olursa, sadece bir tek hit kaydı döndürülür.

void glInitNames(void);

isimler yığını siler(tamamen boşaltır)

void **glPushName**(GLuint **name**);

name ile belirtilen ismi yığına atar. Dolmuş bir yığına isim eklemek GL_STACK_OVERFLOW hatasını oluşturur. İsim yığınlarının derinliği OpenGL uyarlamalarında değişik olabilir ama en azında 64 ismi alabilecek kapasitede olmalıdırlar. İsim yığınının derinliğini öğrenebilmek için **glGetIntegerv()** fonksiyonunu GL_NAME_STACK_DEPTH parametresi ile çağırmalısınız.

void **glPopName**(void);

İsimler yığınının en üstündeki elemanı çıkar. Boş bir yığına uygulana bu fonksiyon GL_STACK_UNDERFLOW hatasını oluşturur.

void **glLoadName**(GLuint **name**);

isimler yığınının en üst elemanını **name** ile belirtilen eleman ile değiştir. Eğer yığın boş ise ki **glInitNames()** fonksiyonundan sonra bu doğrudur, **glLoadName()** GL_INVALID_OPERATION hatasını oluşturur. Bu hatadan kaçınmak için eğer yığın başlangıçta boş ise en azından **glPushName()** fonksiyonu bir kere çağırılmalı ve daha sonra **glLoadName()** fonksiyonu çağırılmalıdır.

Eğer seçme modunda değilseniz **glPushName()**, **glPopName()** ve **glLoadName()** çağrılarına görmezlikten gelinir ve işlenmezler

Hit Kayıtları

Seçme modunda , görüntüleme hacmi ile kesişen primitifler bir "seçme hit" oluşturur(hit-isabetli vuruş demektir). Eğer yığın yönetme fonksiyonları çağırılmış veya **glRenderMode()** çalıştırılmış ise, OpenGL hit kaydını seçme dizisine yazar. Bu süreçle aynı ismi paylaşan objeler –örneğin , birden fazla primitiften oluşan objeler – her primitif için ayrı hit değil,aynı isimleri taşıyan bütün primitifler için tek hit kaydı oluşturur. Hit kayıtlarının diziye yazılmaları **glRenderMode()** fonksiyonu çağırılana kadar garanti değildir,bazı hit kayıtları kaybolabilir.

Primitifler gibi **glRasterPos()** tarafından oluşturulan geçerli koordinatlar da seçme hit'i oluşturabilir. Her hit kaydı 4 bileşenden oluşur :

Hit oluştuğu anda, isimler yığınındaki isimlerin numaraları

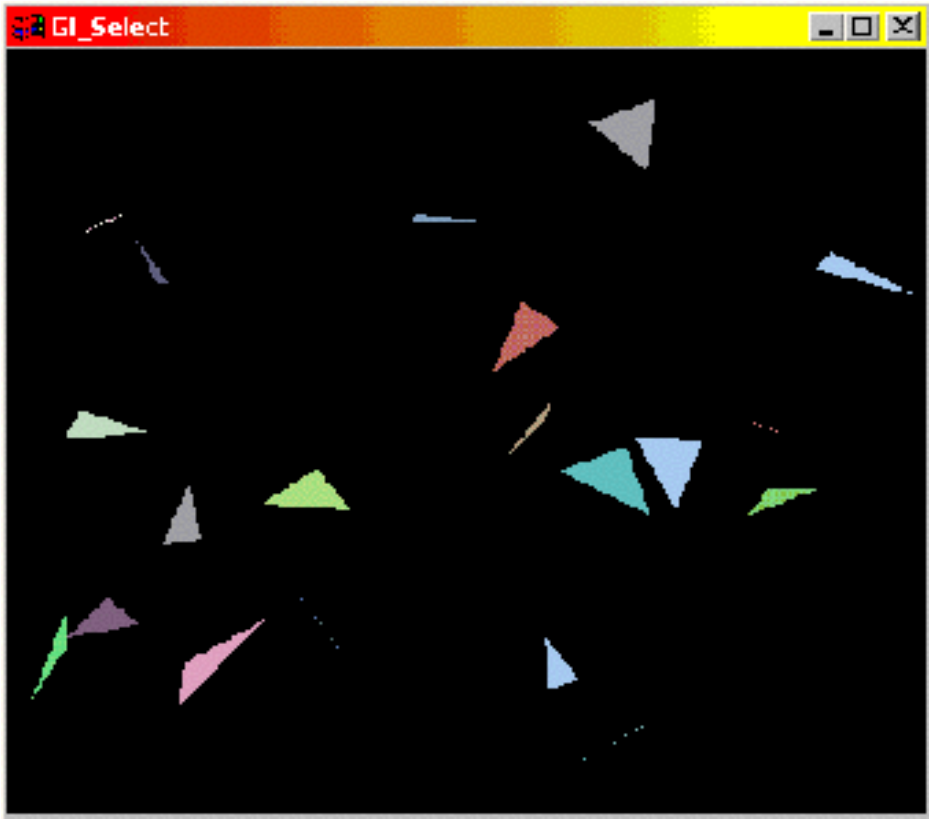
Görüntüleme hacmi ile kesişen bütün primitiflerin bütün vertisleninin pencere koordinatları cinsinden minimum ve maksimum z-değerleri. [0,1] aralığında olan bu iki değerlerden her biri 232-1 ile çarpılarak en yakın işaretsiz tam sayıya dönüştürülür.

Hit anında ,isimler yığınının en alttaki elemanı en başta olacak şekilde tüm içeriği.

Seçme moduna girdiğimiz anda , OpenGL , seçme dizisine bir pointer atar. Hit kaydının diziye her yazılışında , pointer da buna göre dğişmektedir. Eğer yazılan hit kaydı , **glSelectBuffer()** fonksiyonunun **size** argümanı ile belirtilen sayıdan aşarsa , OpenGL yazabildiği kadar hit kayıtlarını yazar ve overflow flagını set eder. Seçme modundan **glRenderMode()** fonksiyonu ile çıkıldığında bu komut yazılan hit kayıtlarının sayısını döndürür, isimler yığınınını temizler,overflow flagını resetler ve yığın pointerini resetler. Eğer overflow flagı set edilmiş ise -1 değeri döndürülür.

Seçme Örneği

Örnekte Seçme modunda üçgenler çizilmekte ve uygun hit kayıtları işlenmeketedir. İlk üçgen hit üretmekte,ikinci üçgen üretmemekte, 3. ve 4. üçgenler de tek bir hit üretmektedirler. Rutinler üçgen çizimi için **-drawTriangle()** ve görüntüleme hacmini belirten kabloçerçeve çizimek için **-drawViewVolume()** . **processHits()** rutini seçme dizisini yazdırmaktadır. Son olarak **selectObjects()** fonksiyonu,hit kayıtları oluşturmak için üçgenleri seçme modunda çizmektedir.



Geribesleme

Geribesleme ,seçme modu gibidir ama farkı şu: seçmede pikseller üretilirken geribeslemede üretilmez,sadece ekran dondurulur,çizim olmaz. Çizim yerine primitiflerin renderlenme biligileri uygulamaya geri gönderilir. Seçme ve geribesleme modları arasındak ana fark da ne tür bilginin döndürüldüğüdür.Seçme modunda atanmış isimler tamsayılar dizisine döndürülür. Geribesleme modunda ise , transform olmuş primitifler hakkında bilgi kayan noktalı sayılar dizisine döndürülür. Geri döndürülüne değerler ne tür primitifinin (nokta, çizgi, çokgen,resim veya bitmap) ,vertex,renk ve diğer primitif bilgilerini belirten token'lardan oluşur. Geribesleme modu **glRenderMode()** fonksiyonunu GL_FEEDBACK parametresi ile çağırılarak başlangıç duruma getirilmiş olur.

Geribesleme moduna girme ve çıkma adımları :

Geribesleme bilgisini tutacak dizi **glFeedbackBuffer()** fonksiyonu ile belirlenir. Fonksiyon argümanları ne tür veri ve diziye ne kadar yazılacağını belirlerler.

glRenderMode() fonksiyonunu GL_FEEDBACK argümanı ile çağırarak geribesleme moduna giriniz. Bu noktadan sonra geribesleme modundan çıkılana kadar , primitifler piksel üretmek için rasterize edilmezler ve framebuffer'in içeriği değişmez.

Primitiflerinizi çiziniz.

glRenderMode() fonksiyonunu GL_RENDER argümanı ile çağırarak geribesleme modundan normal çizim moduna dönebilirsiniz. **glRenderMode()** fonksiyonu taatından döndürülen tamsayı, geribesleme dizisinde tutulan değerlerin sayısını ifade eder.

Geribesleme dizisindeki bilgileri analiz edin.

void **glFeedbackBuffer** (GLsizei **size**, GLenum **type,** GLfloat ***buffer**);

Geribesleme verisi için bir buffer tahsis eder: **buffer** verinin tutulduğu diziye bir pointerdir. **size** argümanı dizide tutulabilecek maksimum değer sayısını ifade eder. **type** argümanı geribesleme dizisindeki her vertex için döndürülen bilginin tipini belirler; bu argümanın alabileceği değerler ve anlamları Tablo 8-1 deki gibidir . **glFeedbackBuffer()** fonksiyonu geribesleme moduna girilmeden önce çağırılmalıdır. Tabloda ki *k* değeri color-index modu için 1 ve RGBA modu için ise 4 tür.

Geribesleme Dizisi

Geribesleme modunda ,rasterize edilecek her primitif (glDrawPixels() veya glCopyPixels() fonksiyonlarına yapılan her çağırı) bir değerler bloğu oluşturur ve bu değerler geribesleme dizisine kopyalanır. Değer sayısı glFeedbackBuffer() fonksiyonunun type argümanı ile belirlenir. Çizmekte olduğunuz primitif tipine uygun bir değer seçmelisiniz : GL_2D veya GL_3D aydınlatılmamış iki veya üç-boyutlu primitifler için, GL_3D_COLOR aydınlatılmış üç-boyutlu primitifler için ve GL_3D_COLOR_TEXTURE veya GL_4D_COLOR_TEXTURE aydınlatılmış, texture uygulanmış , 3 veya 4-boyutlu primitifler için.

Geribesleme değerlerinin her bloğu primitif tipini belirleyen bir kod ile başlar, sonraki bilgiler ise primitif vertislerini tanımlayan bilgiler ve ve bunlarla ilgili veri gelir. Geribesleme tarafından döndürülen *x, y, z* koordinatları pencere koordinatlarındadır; eğer *w* de döndürüldüyle,clip(kesme) koordinatlarındadır.

Kaynak: Bilgisayar ile Grafik İşlemeye Giriş Sunum Raporu, Enes Emre Peçenek