# Amiga Gerçek Zamanlı 3D Grafikler

### Andrew Tyler sürüm 5.4.1,1992

Kaynak: https://amigasourcecodepreservation.gitlab.io/amiga-real-time-3d-graphics/

**İçindekiler**

* Lisans
* Önsöz
* 1. Genel Bakış
  + 1.1. Yeni Bir Ortam
    - 1.1.1. Bu Sanat mı, Yoksa Nedir?
  + 1.2. 16-bit Micro ile Neler Yapabilirsiniz?
  + 1.3. Hız İçin Birleştirildi
  + 1.4. 16-bit Mikroişlemci için Yazma
  + 1.5. Programlar
  + 1.6. Amiga İşletim Sistemi
* 2. 3B Dünya Modelleme
  + 2.1. 3-D Modelleme
  + 2.2. Dönüşümler ve Referans Çerçeveleri
    - 2.2.1. Nesne Çerçevesi
    - 2.2.2. Dünya Çerçevesi
    - 2.2.3. Görünüm Çerçevesi
    - 2.2.4. Ekran
  + 2.3. Koordinat Sistemleri
  + 2.4. Vektörler ve Matrisler
    - 2.4.1. Vektörler
  + 2.5. Veri Yapıları
    - 2.5.1. Değişkenler ve Etiketler
    - 2.5.2. Listeler
  + 2.6. Özet
* 3. Ekranda Çizim
  + 3.1. Ekran
    - 3.1.1. Oyun Alanları ve Sprite'lar
    - 3.1.2. Bit-Düzlemleri ve Renk
    - 3.1.3. Bakır Listeleri
  + 3.2. Blitter ve Ekran
    - 3.2.1. Bit Planes Düzeni — Genel Bakış
  + 3.3. Çizim
    - 3.3.1. Bresenham Çizgi Çizim Algoritması
    - 3.3.2. Bresenham'ı Poligon Dolgusuna Göre Uyarlama
  + 3.4. Örnek Program
    - 3.4.1. *Polydraw'lar*
    - 3.4.2.sistem\_00.s*​*
      * Kütüphaneler
      * Çoklu görev
      * DMA
      * Oyun Alanı
      * Renkler
    - 3.4.3. *çekirdek\_00.s*
      * Anahattı Çizmek
      * Blitter ile Doldurma
      * Blitter ile Kopyalama
    - 3.4.4. *eşitler.s*
    - 3.4.5. *bss\_00.s*
    - 3.4.6. *data\_00.s*
* 4. Pencereleme
  + 4.1. Sutherland-Hodgman Kırpma Algoritması.
  + 4.2. Örnek Program
    - 4.2.1.klip.çerçevesi.s
    - 4.2.2. çekirdek\_01.s
    - 4.2.3. bss\_01.s
* 5. Olayları Perspektif İçerisine Yerleştirmek
  + 5.1. Perspektif Dönüşümü
  + 5.2. Homojen Koordinatlar
  + 5.3. Örnek program
    - 5.3.1. perspektifler
    - 5.3.2. data\_01.s
    - 5.3.3. data\_02.s
    - 5.3.4. bss\_02.s
    - 5.3.5. çekirdek\_02.s
* 6. Basit Rotasyonlar
  + 6.1. Geometrik Dönüşümler
  + 6.2. Ana Eksenler Etrafında Dönmeler
    - 6.2.1. X ekseni etrafında dönüş
    - 6.2.2. Y ekseni etrafında dönüş
    - 6.2.3. Z ekseni etrafında dönüş
    - 6.2.4. Bileşik Dönmeler
  + 6.3. Nesneden Dünyaya Dönüşüm
  + 6.4. Örnek Program
    - 6.4.1. otranw.s
    - 6.4.2. data\_03.s
    - 6.4.3. çekirdek\_03.s
    - 6.4.4. bss\_03.s
* 7. Gizli Yüzeyler ve aydınlatma
  + 7.1. Gizli Yüzey Kaldırma
  + 7.2. Yüzey Normal Birim Vektörünün Hesaplanması
  + 7.3. Aydınlatma ve Renk
    - 7.3.1. Renk Tablosu
  + 7.4. Örnek Programlar
    - 7.4.1. illhide.s
    - 7.4.2. çekirdek\_04.s
    - 7.4.3. veri\_04.s
    - 7.4.4.bss\_04.s
* 8. 3D'de Genel Dönüşümler
  + 8.1. Geometrik Dönüşümler
    - 8.1.1. Rotasyonlar
    - 8.1.2. Ölçeklendirme
    - 8.1.3. Kesme
  + 8.2. Örnek Dönüşümleri
  + 8.3. Fiziksel Gerçekçilik
  + 8.4. Klavye ve Joystick'ten Giriş
    - 8.4.1. Klavye
    - 8.4.2. Joystick
  + 8.5. Örnek Program
    - 8.5.1. transfms.s
    - 8.5.2. çekirdek\_05.s
    - 8.5.3. bss\_05.s
    - 8.5.4. data\_05.s
* 9. Dünya Çapında Uçmak
  + 9.1. Giriş
  + 9.2. Koordinat Dönüşümleri ve Yön Kosinüsleri
  + 9.3. Taban Vektörleri ve Yön Kosinüsleri
  + 9.4. Taban Vektörlerinin Döndürülmesi: Keyfi Bir Eksen Etrafında Dönme
  + 9.5. Hataların Biriktirilmesi
  + 9.6. 3D'de kırpma
  + 9.7. Gözlemcinin Hızı
  + 9.8. Örnek Programlar
    - 9.8.1.wrld\_vw.s
    - 9.8.2. çekirdek\_06.s
    - 9.8.3. bss\_06.s
* 10. Bir Dünya Sahnesi
  + 10.1. Bir Veritabanı
    - 10.1.1. Bir Harita
  + 10.2. Sıralama
    - 10.2.1. Bir Kabarcık Sıralaması
    - 10.2.2. Görüntüleme Dönüşümü
    - 10.2.3. Kontrol Matrisleri
    - 10.2.4. Euler Açıları
  + 10.3. Çalışma Süreleri
  + 10.4. Örnek Program
    - 10.4.1. wrld\_scn.s ve eulr\_scn.s
    - 10.4.2. veri\_06.s
    - 10.4.3. veri\_07.s
    - 10.4.4. veri\_08.s
    - 10.4.5. çekirdek\_07.s
    - 10.4.6. bss\_07.s
    - 10.4.7. sistem\_Ol.s
    - 10.4.8. çekirdek O8.s
  + 10.5. Sonsöz
* Ek A: 68000 Talimat Seti
  + A.1. Kayıtlar
  + A.2. Adresleme Modları
    - A.2.1. Adresleme Modları
  + A.3. Talimat Seti
    - A.3.1. Durum Kodları
  + A.4. Talimat Türlerinin Çeşitleri
* Ek B: Devpac Montajcısı
  + B.1. GenAm2
    - B.1.1. Editör
    - B.1.2. Dosyada Hareket Etme
    - B.1.3. Metin Düzenleme
    - B.1.4. Metin Hareketi
    - B.1.5. Montaj
    - B.1.6. Seçenekler
    - B.1.7. Yönergeler
    - B.1.8. Hata ayıklama
    - B.1.9. Programları Çalıştırma
    - B.1.10. Kesme Noktaları
    - B.1.11. Çeşitli
    - B.1.12. Böcek Avı
* Ek C: Sayı Sistemleri
  + C.1. İkili
  + C.2. Hexadecimal (kısaca hex)
  + C.3. Negatif Sayılar
* Ek D: Yonga Kayıtları
* Ek E: Vektörler ve Matrisler
  + E.1. Vektörler
  + E.2. Matrisler
  + E.3. Vektör Ürünleri
    - E.3.1. Skaler (Nokta) Ürün
    - E.3.2. Vektör (Çapraz) Ürün
    - E.3.3. Yüzey Normal Vektörleri
    - E.3.4. Temel Vektörler
  + E.4. Matrisler
  + E.5. Homojen Koordinatlar
* Ek F: Geometrik ve Koordinat Dönüşümleri
  + F.1. Koordinat Sistemleri ve Referans Çerçeveleri
* Ek G: Program Yapısı
  + G.1. Bölüm 3
  + G.2. Bölüm 4
  + G.3. Bölüm 5
  + G.4. Bölüm 6
  + G.5. Bölüm 7
  + G.6. Bölüm 8
  + G.7. Bölüm 9
  + G.8. Bölüm 10

Önsöz

"Bir resim bin kelimeye bedeldir". Bu ifade her şeyi özetliyor.

Birkaç yıl önce, bilgisayar grafikleri hakkında bir kitap açtığımda, bilgisayarlar tarafından üretilen gerçekçi nesnelerin güzel simülasyonları karşısında şaşkına dönmüştüm. Ancak bunlar, o zamanın popüler kişisel mikro bilgisayarlarından çok daha güçlü olan ve neredeyse sadece 8 bitlik olan son teknoloji makinelerden geliyordu.

16 bitlik mikroların gelişiyle birlikte işler belirgin şekilde değişti. Ekstra güçleri ve bellekleri, boyama programlarından katı 3B ilkel nesnelere (nesnelere) sahip hızlı uçuş simülatörlerine kadar tüm grafik uygulamaları üzerinde anında bir etki yarattı. Commodore Amiga gibi mikroların düşük fiyatı ve yüksek gücü, herkesin nispeten düşük bir maliyetle yüksek kaliteli bilgisayar grafiklerinin (özellikle oyunlarda) keyfini çıkarabileceği anlamına geliyordu. Ancak başkalarının programlarının keyfini çıkarmak eğlencenin yalnızca yarısıdır. Şaşırtıcı bir şekilde, bunları yazmak göründüğü kadar zor değildir. Elbette yol boyunca öğrenilecek oldukça fazla teknoloji vardır, ancak dramatik etkinin büyük bir kısmı, oldukça standart algoritmaları çok hızlı gerçekleştiren makinelerin hızından kaynaklanmaktadır.

Grafik programlamaya ilk ilgi duyduğumda ve makine kodunda mümkün olduğunca hızlı olmasını istediğimde, bana literatürde temel bilgilerin ince bir şekilde yayılmış olduğu göründü. Makine kodu programlama ve bilgisayar grafikleri hakkında kesinlikle kitaplar vardı; hatta makine kodu grafik programlama hakkında birkaç kitap bile vardı. Ama bir şekilde aradığım dengeyi hiçbir zaman bulamadım. Bilgisayar grafikleri hakkındaki standart metinler, özellikle bir sahneyi keyfi bir bakış açısından nasıl resmedeceğiniz konusunda dönüşümlerin belirli yönleri hakkında inanılmaz derecede belirsiz görünüyordu. Belki de oldukça mantıksız bir şekilde, her şeyi profesyonel bir gizem perdesinin arkasına saklama eğilimi olduğunu hissettim; kesinlikle vektörlerin ve matrislerin matematiğini anlamama önemli ölçüde yardımcı oldu, ancak bunların hepsi yıllar önce çözülmüştü ve oldukça basit olmalıydı? Belki de sadece bendim! Neyse, gerçek zamanlı çalışacak (bir uçuş simülatörü gibi) 3B katı grafik programları yazmak istiyordum ve bana bunu nasıl yapacağımı söyleyecek kimseyi bulamadım. Ticari oyunlar yazan kişiler kesinlikle biliyordu, ancak açık nedenlerden dolayı söylemiyorlardı! Dergilerde çok faydalı birkaç tefrika makale vardı ama, eminim ki zorunluluktan dolayı, bunlar çoğunlukla çok kısaydı ve tam olarak istediğim gibi değildi.

İşler, İleri Mikrobilgisayar Yazılımı (başka bir deyişle "68000'de Montaj Dili Programlama" anlamına geliyordu) üzerine bir üniversite dersi vermem istendiğinde doruk noktasına ulaştı. Özellikle montaj dilinde programlama öğretmek, uygulama ilgi çekici olmadığı sürece çok kısır bir uğraş olabilir. Grafiklerden daha iyi bir uygulama ve Amiga'dan (fiyatına göre) daha iyi bir makine olabilir mi?

Bu kitap, bilgisayar grafikleri dünyasına girme ve bazı temel bilgileri uzman olmayanlar için anlaşılır kılma (umarım) çabalarımın bir sonucu olarak ortaya çıktı. Assembly dilinde hızlı 3D (sözde vektör) grafiklerle ilgilidir. Bu kitaptaki programların kendi türlerinin en verimli, en zarif ve en hızlısı olduğunun kesinlikle bir garantisi yoktur. Ancak makul derecede hızlıdırlar. Kesinlikle bazı ticari programlar kadar hızlıdırlar! Zeki okuyucu şüphesiz iyileştirmeler yapabilecektir (ve umarım bana söyleyebilir).

Okuyucunun aşağıdaki konulardan herhangi biri hakkında önceden bilgi sahibi olduğu varsayılmamaktadır; bunların hepsi grafik sürecinde önemli bir rol oynar: Amiga İşletim Sistemi, vektörler ve matrisler. Eklerde daha fazla açıklama bulunmaktadır. Bu, kitabın bu konular hakkında kapsamlı tartışmalar içerdiği anlamına gelmez, yalnızca eldeki amaç için yeterlidir. Meraklısı şüphesiz bunlara eklemeler yapmak isteyecektir.

Montaj dili açısından, bir Ek, ​​talimat setinin bir listesini ve (en önemlisi) adresleme modlarını içermesine rağmen, olup biteni tam olarak anlamak isteyen okuyucunun elinde 68000 kodlu bir referans kitabının olacağı varsayılmaktadır (bunlar cep boyutunda çok ucuza temin edilebilir).

Kitaptaki programların yazılması, derlenmesi, hata ayıklaması ve çalıştırılması için, birkaç iyi ticari derleyici/hata ayıklayıcıdan biri olan Hisoft'un güçlü ve kullanıcı dostu Devpac Amiga derleyicisi kullanılmıştır. Bu, tüm işlevlerin gerçekleştirilebileceği entegre bir paket olarak gelir. Derleyici hakkında daha fazla bilgi Ek 2'de verilmiştir .

Kitap seri halinde düzenlenmiştir. Her bölüm farklı bir konuyu ele alır ve örnek programlarla uygulamasını gösterir. Deneyimli okuyucuya ilk bölümler yaya gibi görünecektir. Yeni başlayanlara ise öyle görünmeyecektir. Genel sürece dair gerçekten kolay bir giriş yoktur ve bu nedenle her aşama (biraz yapay bir bölüm) ayrıntılı olarak ayrı ayrı ele alınır. Grafik "boru hattının" her aşaması belirli bir görevi yerine getirir ve kendi algoritması ve stratejisine sahiptir. Bölümler, genel sürecin oluşumunu yansıtacak şekilde düzenlenmiştir. Her bölümün kendi örnek programları vardır ve önceki bölümlerden kaydedilen programlar sonraki bölümlerde kullanılır, böylece birden fazla kez girilmeleri gerekmez. Bu şekilde kitabın sonundaki örnek programlar en büyük ve en karmaşık olanlar olur, ancak her yeni bölüm için girmeniz gereken kod miktarı gerçekten çok fazla artmaz. Programlar Amiga için yazılmıştır ancak ekran ve işletim sistemiyle ilgili bazı ayrıntılar dışında grafik rutinleri tamamen bağımsız ve kendi kendine yeterli olduğundan, 680000 tabanlı herhangi bir bilgisayarda çalışacak şekilde değiştirilebilir.

Bilgisayar grafikleri çok geniş bir konudur; bu uzunluktaki bir kitap yalnızca küçük bir kısmını kapsayabilir. Özellikle de yalnızca tanımlayıcı olmayıp çalışan programlar içerdiği için. Ray Tracing ve Radiosity yöntemleri gibi teknikler belki de gelecekteki, daha güçlü bir kişisel bilgisayar nesline daha uygundur. Ancak bu gelecek; burada tartışılan yazılım rutinlerinin çoğunun gelecekteki makinelerde donanım "geometri motorları" ile değiştirilmesi muhtemeldir.

O zamana kadar 3D grafiklerin "baytlara çarparak" yapılması gerekecek.

Son olarak çok önemli bir uyarı. Deneyimli programcı her şeyi bilir.

Çalışmanızın sık sık yedeklerini alın — yaklaşık her iki saatte bir. Bu tür programları donanıma yakın bir şekilde yazarken, güvenlik ağı yoktur! Arızalı bir program sistemi kolayca çökertebilir ve disk sürücüsü çökerken çöpleri diske püskürtebilir. Benim başıma geldi. Son çalışmalarınızdan mümkün olduğunca azını kaybetmek istersiniz. Bununla ilişkili yararlı bir uygulama vardır — yeni bir program çalıştırmadan önce diske yazma koruması koyun. Çökerse diski kurtarmaya yardımcı olur.

İyi şanlar.

*Andrew Tyler* ,1992

1. Genel Bakış

Bilgisayar grafikleri bilgisayar tutkunlarının azınlık ilgi alanı değildir. Milyarlarca dolarlık bir endüstridir. Hollywood'un film yapımına 3 milyar dolar harcadığı 1982'de bile, dünya ticari bilgisayar grafikleri endüstrisi 2 milyar dolar harcamış ve yılda %30 oranında büyümüştür. Aynı yıl ABD'de video oyunlarına 10 milyar dolar harcanmıştır. O zamandan beri bir duraklama olmamıştır. Bilgisayar grafikleri gerçekten çok büyük bir iştir.

Mikrobilgisayar sahibi, makinesi için en iyi grafiklerden bazılarını oyunlarda bulur, bunların çoğu doğrudan profesyonel bilgisayar dergilerinden alınan gelişmiş konseptleri kullanır. Küçük makineler için, işlemcinin hızı ve RAM boyutu tarafından belirlenen, neyin başarılabileceği konusunda her zaman sınırlamalar vardır. Ancak son yıllarda popüler mikrobilgisayar, çok düşük bir fiyata önemli bir hesaplama gücüne sahip olması ve minimum maliyetle karmaşık grafikler sağlaması nedeniyle paraya fazlasıyla değer katmıştır. Amiga tam da böyle bir bilgisayardır. Bilgisayar donanımının güç/fiyat oranındaki bu patlama, popüler mikrobilgisayar sahiplerinin eline muazzam bir hesaplama yeteneği vermiş ve profesyonellerin alanı olan gelişmiş grafik tekniklerini herkesin kullanımına sunmuştur.

Bu kitabın amacı gerçek zamanlı çalışan ve pencereleme (kırpma), gizli yüzey kaldırma, ışık kaynağından aydınlatma, joystick kontrolü, tam perspektif ve dönme dönüşümleri gibi özellikler içeren ve bir uçuş simülatörü tipi programla sonuçlanan hızlı 3B katı grafik rutinleri geliştirmektir. Programlar bir Amiga 500'de çalışmak üzere 68000 makine kodunda yazılmıştır ancak algoritmalar herhangi bir makine için geçerlidir. Kısacası, bir uçuş simülatörüne başlamak için gereken her şey.

Programlar maksimum hız için assembly dilinde yazılmıştır ve Hisoft Devpac Amiga derleyicisi kullanılarak test edilmiş ve çalıştırılmıştır. Uygun fiyatlı birçok mükemmel ticari derleyici mevcuttur ve hatta bazıları kamuya açıktır. Bir programda inatçı ve inatçı bir hatayı ararken dost canlısı olmayan bir derleyiciden daha sinir bozucu bir şey yoktur. Devpac derleyicisi, bu kitaptaki programları geliştirmek için gereken uzun saatler boyunca dost canlısı ve yardımsever bir arkadaş olmuştur.

1.1. Yeni Bir Ortam

'Bilgisayar grafikleri' nedir? Kesinlikle bir dereceye kadar gizemlidir. Hala nispeten genç bir konu olduğu için evrimi hızla devam ediyor ve günümüz bilgisayarlarının gücü ve bunlara dahil edilen özel grafik donanımıyla yakından bağlantılı. Dünün birçok sorununa bir zamanlar yazılıma dayalı olan çözümler artık donanımda büyük bir hızla sağlanıyor. Bu kitapta geliştirilen türden yazılımların çoğunun gelecekteki makinelerde özel 'geometri motorları' ile değiştirilmesi muhtemeldir.

1.1.1. Bu Sanat mı, Yoksa Nedir?

İnsanlar karmaşık görsel desenleri oluşturma ve tanımada çok iyidirler ancak aritmetikte çok iyi değillerdir. Buna karşılık, dijital bilgisayarlar ikili aritmetikte mükemmel olacak şekilde tasarlanmıştır. Başka neler yapabilecekleri, temel ikili aritmetikten karmaşık matematiksel fonksiyonların ne kadar iyi oluşturulabileceğine bağlıdır. Burada bir sınırlama vardır çünkü bir bilgisayardaki sayılar, kendilerine atanan bit sayısından daha doğru olamaz ancak bunun dışında, karmaşık matematiksel hesaplamaların çok mütevazı mikrobilgisayarlarda bile hızlı bir şekilde yapılabileceği açıktır.

Bilgisayar grafiklerinde, bilgisayar çizimin matematiği olan geometriyle ilişkili herhangi bir hesaplamaya muazzam bir hız katar. Geometri, çizgiler ve yüzeyler arasındaki kesin matematiksel ilişkilerle ilgilendiğinden, bilgisayarın çalışma şekliyle ideal olarak eşleştirilmiştir. Bilgisayarlarla çizim yapmanın iyi ve kötü haberi şudur: kesin matematiksel işlevler grafiksel olarak ışık hızında ifade edilebilir ancak bunları doğal nesneler gibi göstermek çok daha fazla çalışma gerektirir. Aslında bilgisayar grafiklerindeki çabanın çoğu artık geometrinin mükemmel ama steril görüntülerini insan tüketimine uygun hale getirmek için 'berbat etmekle' ilgilidir. Bunu yapmak bilgisayarlarla daha az, Walt Disney tarafından yıllar önce keşfedilen geleneksel animasyon becerileriyle daha çok ilgilidir.

Bir bilgisayarla kesin matematiksel şekiller çizmek çok kolaydır çünkü bu tür şekiller bir formülden üretilebilir. Bir daire, basit bir matematiksel fonksiyonun örneğidir. Bir xy koordinat sisteminin kökeninde merkezlenmiş bir daire için formül şudur:

x2 + y2 = r2​

Böyle bir fonksiyon bilardo topu için iyi bir başlangıç ​​noktasıdır ancak elma için kötü bir başlangıç ​​noktasıdır, ancak yüzeysel olarak fark o kadar da büyük değildir (her ikisi de parlak bir dış yüzeye sahip genel olarak küresel bir şekle sahiptir). Bir elmayı çizmek için bir bilgisayarı nasıl kullanabileceğimizi düşünelim.

Öncelikle iyi bir başlangıç ​​noktası olmalı. Elma için matematiksel bir formül diye bir şey yoktur. Tüm elmalar farklıdır. Ancak elmaların tipik bir şekli vardır ve insan sanatçı bunu deneyimlerinden bilir. Ancak bir sanatçı natürmorttaki tüm elmaları aynı şekilde çizmez, çok sıkıcı olurdu. Tekrarı ve basitliği önlemek için bir bilgisayarı programlamak zordur.

Elmaları çizmenin bir yolu, elma şekline sahip eğri denklemlerini kullanmaktır. x, y ve z'nin yüksek kuvvetlerine sahip fonksiyonlar seçerek, istenildiği kadar keskinlik veya düzlük dahil edilebilir. Bu, bikübik yamalar, Bezier fonksiyonları ve beta-spline'ların dünyasıdır. Bu kesinlikle çeşitliliğe izin verir, ancak önemli bir hesaplama çabasıyla. Bunu yapmanın bir yolu, farklı elma ana hatlarını bir veri tabanında (x,y) koordinat çiftleri olarak tutmak ve ardından bunları bir "nokta birleştirme" resminde olduğu gibi birleştirmek için eğri ve yüzey uydurma tekniklerini kullanmaktır. Katı yüzey modellemesinin erken gelişiminde bir prototip olan Martin Newell'in ünlü çaydanlığı bu şekilde inşa edilmiştir. Teknik dilde, üç Bezier eğrisinden oluşan bir ana hattan inşa edilebilir. Çaydanlık simetrik olduğundan, yüzeyi (ağızlık hariç) daha sonra ana hat merkezi dikey eksen etrafında döndürülerek oluşturulur.

Başka bir yol da eğrilerden tamamen kaçınmak ve bunun yerine elmanın yüzeyini bir mücevher taşı gibi birçok düz yüzeye bölmektir. Düz ve çok kenarlı olan küçük yüzeyler çokgenlerdir ve elmanın yüzeyi bir çokgen ağıdır. Bu yaklaşım, eğri yamalar kullanmaktan daha az zaman alır ancak çokgenler arasındaki keskin sınır kenarlarını gizleme sorunu devam eder.

Bu, ikna edici bir görüntü üretmede bir sonraki incelik düzeyine yol açar. Matematiksel bir fonksiyon kendi başına fizik yasalarından habersizdir. Bunlar bize o kadar tanıdıktır ki bunları hafife alırız: cam şeffaftır ancak ahşap opak, metaller parlak ve ışıl ışıl görünür ancak insan cildi donuk ve dağınıktır. Bir şekilde bu ince ancak temel ipuçları dahil edilmelidir. En önemli ilk adım, opak nesnelerin arka yüzeylerini görünmez hale getirmektir. Buna gizli yüzey kaldırma denir ve görevin görünürdeki basitliğine rağmen oldukça zor olduğu ortaya çıkar. Bunu yapmanın etkili ve kapsamlı yollarını araştırmak için çok zaman harcanmıştır. Sonra yüzey yapısına dair görsel ipuçları olmalıdır. Açık bir adım, bir tarafı diğerinden daha parlak olacak şekilde onu bir ışık kaynağıyla aydınlatmaktır.

Bir sonraki rafine etme seviyesinde, yüzeyin gerçek görünmesi için "doğal" bir şekilde dokulu ve desenli olması gerekir. Bu konuda programcıya Benoit Mandelbrot tarafından geliştirilen ve desteklenen fraktalların matematiği yardımcı olur. Bu, kendi kendine benzer yapıların geometrisidir ve yapıların mükemmel çizgilerden ve yüzeylerden inşa edildiği Öklid geometrisinden oldukça farklıdır. Doğal nesneler kendi kendine benzer yapılarla çok fazla ortak noktaya sahip gibi görünür ve benzerlik tam olmasa bile, bunlar onlar tarafından ikna edici bir şekilde modellenmiştir. Kendi kendine benzer bir yapı, herhangi bir büyütme seviyesinde aynı görünüme sahip olan bir yapıdır. Elbette doğal nesneler bu tanımı yalnızca sınırlı bir boyut aralığında karşılayabilir ancak genellikle çok ikna edici sonuçlar üretir. Örneğin, bir eğrelti otunun yan dalı büyütüldüğünde ana dala benzer görünür ve büyütme altındaki küçük çakıllar kayalara benzer. Doğa bu tür yapılarla doludur. Ek bir bonus olarak, kendi kendine benzer yapıların ve manzaraların nispeten küçük miktardaki bilgiden oluşturulmasına izin veren algoritmalar keşfedilmiştir. Bu, programcıyı karmaşık bir sahnenin her bir ayrıntısını oluşturmak için devasa bir veri tabanı taşımaktan kurtarır.

İkna edici bir görüntü elde etmek için tüm bu adımlar gereklidir. Bir görüntüyü gerçekçi göstermek için bu kadar çok görsel zenginliğin gerekli olması, insanların çok gelişmiş desen tanıma yeteneğini test eder.

Bütün bunlar yapıldığında, elimizde ne kalır? Sadece bir elmayı boyamanın çok dolambaçlı bir yolu mu? Aradaki fark, yazılımda bir kez yaratıldığında grafik varlığın bağımsız bir varoluşa sahip olmasıdır. Ekrandaki resim sadece son aşamadır. Şu anda görüntülenmese bile, programda yer alan kurallara göre gelişebilir. Gerçek yaşamdan modellenen nesneler yaratma konusunda bile bir kısıtlama yoktur. Bilgisayarın içinde yeni "yaşam formları" icat etmek mümkündür. Bilgisayar Destekli Tasarım'da (CAD) her zaman olan budur. Makineler, maddi nesneler olarak var olmadan çok önce bilgisayarın içinde tasarlanır, inşa edilir ve test edilir. Simülatörlerde ve oyunlarda bu yön mümkün olduğunca ileri götürülür. Bilgisayar oyunları yapay gerçeklikler üretmede uzmanlaşır; ne kadar egzotik olursa o kadar iyidir.

Giriş-çıkış aygıtlarındaki gelecekteki gelişmeler şüphesiz şu anda bilgisayar grafikleri olarak adlandırılan şey üzerinde büyük bir etkiye sahip olacaktır. Şu anda vurgu gerçekçi görüntüler üretmek üzerinedir. Ancak görüntüler yalnızca gözler aracılığıyla insan girdisi için tasarlanmış bilgisayar çıktısıdır. Tüm duyular dahil olduğunda buna ne ad verilecek? Zaten her göze ayrı girdi veren gözlüklerin ve ellerde dokunsal uyarımın yardımıyla, bilgisayarın içindeki dünyaya tamamen girmek mümkündür. Bu Sanal Gerçeklik veya Siber Uzaydır. Bilgisayar, ara bir arayüze ihtiyaç duymadan doğrudan insan sinir sistemine bağlandığında nasıl olacak? Vücudu beslemenin küçük bir düşüncesi dışında, bilgisayarın içinde tamamen yapay bir varoluş yaşamak mümkün olacaktır.

Bilgisayar grafikleri, bilgisayarların ilk kez insan girdisine yanıt olarak görsel bir çıktı ürettiği zaman başlayan çok uzun bir kamanın ince ucudur. Nerede biteceği bilinmiyor, ancak yol boyunca çok eğlenceli olacağı kesin.

1.2. 16-bit Micro ile Neler Yapabilirsiniz?

Bu sorunun cevabı, neyin başarılabilir olduğuna ve iyi bir örneği Lucasfilm Ltd. tarafından geliştirilen ve şu anda Pixar tarafından kullanılan Reyes sistemi olan güçlü bir ticari sisteme bakarak en iyi şekilde gösterilebilir. Bu, "Andre ve Wally B'nin Maceraları", "Luxo Jr.", "Red's Dream" ve "Genç Sherlock Holmes" filmindeki animasyon şövalye dizisi de dahil olmak üzere bir dizi iyi bilinen kısa film sekansı yapmak için kullanılmıştır. Reyes sistemi, gerçek yaşam kadar görsel olarak zengin grafikler içeren tam uzunlukta bir uzun metrajlı filmi yaklaşık bir yılda hesaplamak için kurulmuştur. Bir sinema filminin yaklaşık 2 saat sürdüğünü ve filmin saniyede 24 kare hızında çalıştığını varsayarsak, bu, her karenin yaklaşık üç dakikada hesaplanması (işlenmesi) gerektiği anlamına gelir.

Bu sistemdeki temel strateji, bir sahnedeki her nesneyi (geometrik ilkel) bir pikselin 1/4'ü (ekranda görülebilen en küçük birim) alanı olan alt piksel boyutunda dörtgenler olan mikro poligonlardan oluşan bir ağ ile temsil etmektir. Tüm gölgelendirme ve görünürlük hesaplamaları bu mikro poligonlar üzerinde yapılır. Genel resim, yalnızca görünür kısımların gerçekten çizildiği bir film seti gibi oluşturulur. Mikro poligonlar, belirli bir görüş açısının dışında yer alırlarsa veya çok yakın veya çok uzak olurlarsa görünmez kabul edilirler. Son sistem, hareket bulanıklığı gibi incelikleri içerir; hareket halindeki nesnelerin arka kenarlarının bulanık görünmesine neden olan etki. Bu, hareket izlenimini artırmak için kullanılan araçlardan biridir ve geleneksel karikatüristlerden öğrenilen bir başka derstir.

Bu sistemdeki oldukça karmaşık bir resim, 1024x612 piksel çözünürlüklü bir sahneyi işlemek için genellikle 7 milyondan biraz daha az mikropoligon kullanır. 4 ışık kaynağı ve 15 doku kanalıyla bir resim, VAX11/780'den 4-6 kat daha hızlı olan bir CCI 6/32 bilgisayarında hesaplanması yaklaşık 8 saat CPU süresi alır. "Young Sherlock Holmes" filmindeki kareler aynı çözünürlükteydi ve hesaplanması kare başına bir saat sürdü. Son filmde depolanan tüm kareler geleneksel bir filmde olduğu gibi oynatılır.

Ancak yüksek kaliteli resimler üretmek için bu kadar uzağa gitmek gerekli değildir. Ölümlülerin neredeyse ulaşabileceği fiyatlarda çok sayıda "kişisel" grafik istasyonu mevcuttur. Silicon Graphics tarafından üretilen Personal Iris makineleri iyi birer örnektir. 4096'lık bir paletten 256 renk (8 düzlem) sunarlar ve bir donanım "geometri motoru" kullanarak ölçekleme, döndürme, gizli çizgi kaldırma ve aydınlatma gibi dönüşümleri gerçekleştirerek gerçek zamanlı olarak 3B hareket üretebilirler. CPU, R3010 FPU'ya (kayan nokta birimi) sahip 20MHz R3000 RISC işlemcidir. Burada hızı en üst düzeye çıkarmak için RISC teknolojisi kullanılmıştır, ancak 1986'dan önce Silicon Graphics'in 68000 işlemciyi kullandığını belirtmek ilginçtir. Bu tür makinelerin kişisel bilgisayar pazarına girmesi uzun sürmeyecektir.

Peki ya 512 kbyte RAM ve 7 MHz hızında çalışan bir CPU'ya sahip Amiga 500 gibi bir mikro ne olacak? Ayrıntılı grafikler için potansiyel biraz daha azdır, özellikle de kareler gerçek zamanlı olarak, tahammül edilemez titremeyi önleyecek kadar hızlı çalışacaksa. Ancak ne kadar çok şey başarılabileceği şaşırtıcıdır. Hız için, poligon ağları kullanarak katı nesneler oluşturmak en çekici olanıdır çünkü yalnızca köşelerin depolanmasını gerektirir ve büyük bir nesne çok az miktarda bilgiyle tanımlanabilir.

Ayrıca, çokgenler düz çizgilerle birleştirilmiş köşe kümeleri olduğundan, dahil olan en karmaşık cebir basit geometrinin cebiri olacaktır. Kullanacağımız strateji budur.

1.3. Hız İçin Birleştirildi

Birçok bilgisayar dili vardır ancak montaj dili, donanıma mümkün olduğunca yakın olma ve eldeki uygulamaya göre uyarlama yapma konusunda en iyi fırsatı verir. Bu kitaptaki tüm programlar 68000 montaj dilinde yazılmıştır ve ilk bölümdeki "ev işleri" ve belirli donanım işlevleri dışında, Amiga işletim sistemindeki rutinlerden hiçbirini kullanmaz. Bu nedenle programlar, en zor şey genel program yapısı olduğundan, 68000 dışındaki bir işlemcide çalışacak şekilde kolayca yeniden yazılabilir. Dil ayrıntıları ikincildir.

Assembly dili, kendine özgü mazoşist bir çekiciliğe sahip, çok titiz ve affedici değildir. Talimatların sözdiziminin ötesinde çok az dilbilgisi yapısına sahiptir ve verimli programlama için ana ölçütler hız, kayıtların ve belleğin ekonomik kullanımı ve verimli parametre geçişidir. Bazen bunlar arasında, özellikle de bellek sıkıntısı olmadığında, çatışmalar olur. Hızın çok önemli olduğu yerlerde, programlar genellikle zaman alıcı alt rutin çağrılarından kaçınmak için kısalıktan ödün verirler.

Bu kitaptaki programlar Hisoft'un Devpac Amiga derleyicisi kullanılarak derlenmiş ve çalıştırılmıştır ancak biçimine uyması için değişiklikler yapıldığı sürece herhangi bir başka derleyici de iş görür. Basit ancak güçlü INCLUDE yönergesi, dosyaların derleme zamanında global değişkenler tanımlamaya gerek kalmadan bir araya getirilmesini sağlar. INCLUDE yönergesi, belleğin izin verdiği herhangi bir derinliğe yerleştirilebilir, böylece her bölüm önceki bölümlerden programları DAHİL edebilir. Bu şekilde neredeyse hiç tekrarlama olmaz ve bir program dosyası girildikten sonra daha sonra kullanılabilir. Bu nedenle genel program kitap ilerledikçe giderek büyür ve pratik olarak hiçbir programlama çabası boşa gitmez. Son program önceki tüm bölümleri DAHİL EDER. Yapılması gereken tek bağlantı budur ve zahmetsizdir.

Ek 2'de genel olarak derleyici kullanımına ve özellikle Devpac derleyicisine ilişkin kısa bir açıklama verilmiş olup, en yararlı olduğu bulunan komutlar da buna dahil edilmiştir.

1.4. 16-bit Mikroişlemci için Yazma

16 bitlik bir mikro için derleyicide program yazmak, 8 bitlik bir mikro için yazmaktan oldukça farklıdır. Mevcut daha güçlü adresleme modlarının yanı sıra, konum bağımlı ve bağımsız kodda somutlaştırılan fikirlere odaklanan temel bir fark vardır. Mutlak ve taşınabilir kod gibi diğer benzer sesli terimler resimi biraz karıştırır. Bunların ne anlama geldiğini tartışacağız çünkü bir programın derleyicide nasıl yazıldığı üzerinde derin bir etkiye sahiptirler.

8 bitlik bir mikroda genellikle bir seferde yalnızca bir program RAM'e ve sabit bir konuma yüklenir. Elbette, CP/M gibi bir işletim sisteminin programların çalışmasını denetlediğinde işler daha karmaşıktır. Ancak yerleşik BASIC ve çok az başka şeye sahip küçük mikrolarda, işletim sistemi değişken alanı için sabit alan ayırır ve diğer her şeyi geçerli program için serbest bırakır. Programın bellekte nerede bulunduğunu bilmek, programcının hayatını kolaylaştırır çünkü değişkenler için sabit adresler atanabilir ve bunlar asla değişmez. Sabit bellek konumlarını doğrudan adresleyen bir programın, konuma bağlı veya mutlak kodda yazıldığı söylenir.

Bu tür kodlar işletim sistemlerine sahip bilgisayarlar için yazılabilmesine rağmen, çok daha fazla esneklik sağlayan ve birden fazla programın aynı anda bellekte bulunmasına izin veren başka bir yol daha vardır. Bunun bir sonucu olarak, belirli bir programın bellekteki gerçek konumu çalışma zamanına kadar bilinmeyecektir. Sonuç olarak, program yüklenip çalıştırılıncaya kadar sabitlenmediği için programda gerçek bir sayısal adrese başvurulamaz.

Bu sorunu aşmanın birkaç yolu vardır. Bir yol, özellikle konumdan bağımsız kod üretmek için tasarlanmış işlemcinin adresleme modunu kullanmaktır. Buna PC (program sayacı) göreli adresleme denir. Yaptığı şey, bir adresi mutlak bir değer olarak değil, referansın yapıldığı program sayacının değerine göre belirlemektir. Derlenmiş kod, işlemciye program sayacının geçerli değerine bir yer değiştirme ekleyerek veya çıkararak gerçek adresi hesaplamasını söyleyecektir; bu değer her zaman programın başlangıcına göre sabit bir değere sahip olacaktır.

Başka bir yol da tüm adresleri bir adres kaydında tutulan bir temel adresten veya işaretçiden hesaplamaktır. Program daha sonra sürekli olarak adres kaydından ofsetlere başvuracaktır ancak program yazılırken adres için gerçek bir değer belirtilmesi gerekmez. Kayıt, elbette, bu şekilde ayrılmışken başka bir şey için kullanılamaz. Özel kayıt, programın başında doğru işaretçi ile ayarlanmalıdır. İyi bir işaretçi, programın sonunun adresidir.

Başka bir yol da derleyicinin her şeyle ilgilenmesine ve yeniden konumlandırılabilir kod üretmesine izin vermektir. Bu, belirli adreslere hiçbir referans yapılmayan, bunun yerine etiketlerin kullanıldığı koddur. Etiket adı, bilgilendirici ve programcıya yardımcı olacak şekilde seçilir. Örneğin, COLOR, bir poligonun geçerli renginin bayt uzunluk değerinin tutulduğu uzun sözcük adresinin etiketi olabilir. Derleyici böyle bir etiketi yeniden konumlandırılabilir olarak işaretler ve adresi, program yüklendiğinde bilgisayar işletim sistemi tarafından nihayet sabitlenir.

Bu kitaptaki tüm programlar Devpac derleyicisi tarafından üretilen taşınabilir kodu kullanır. Yazması basittir.

68000'in talimat seti uzun ve karmaşıktır. Gücünü ve zarafetini tam olarak takdir etmek için okuyucu Motorola 16-Bit Mikroişlemci Kullanıcı Kılavuzu'na başvurmalıdır. Ek 1'de kısa bir liste verilmiştir .

1.5. Programlar

Bu kitaptaki programlar Devpac derleyicisi kullanılarak yazılmıştır ve çalıştırılmaya hazırdır. Bir program girildikten sonra yapılması gereken tek şey onu düzenleyicinin içinden birleştirmektir ve açıklandığı gibi çalışacaktır. Program dosyalarının hepsinin uzantısı .s'dir çünkü kaynak dosyalarıdır. Bir program bağımsız olarak çalışacaksa .prg dosya uzantısıyla diske derlenebilir. Aslında yer darlığı nedeniyle, Bölüm 6'dan sonraki tüm programların diske derlenmesi gerekebilir.

Programların tümü mümkün olduğunca hatasız olduğundan emin olmak için kapsamlı bir şekilde çalıştırıldı ve listeler, hataların girebileceği başka transkripsiyon aşamaları olmadığından emin olmak için PRINT BLOCK olanağı kullanılarak montajcı Editörü içinden elde edildi. Ancak tüm insan çabalarında olduğu gibi, programların tamamen hatasız olduğunun garantisi olamaz.

Programlar şüphesiz ki kendi türlerinin en hızlı veya en zarif örnekleri değiller ancak, vurgunun öğretmeye olduğu bu tür bir eğitimde, asıl mesele işlerin nasıl yapıldığını anlamaktır. Zeki okuyucular bunları geliştirmenin akıllı yollarını çabucak keşfedecektir. Her durumda en iyi ticari programlar tescillidir ve bizden gizli tutulur.

1.6. Amiga İşletim Sistemi

Amiga işletim sistemi büyük ve karmaşıktır ve birçok seviyede çalışır. Giriş seviyesine bağlı olarak aynı şeyi yapmanın birçok yolu vardır.

Cihazdan bağımsız rutinleri kullanmak programların taşınabilir olmasını, yani donanım ayrıntılarından korunmasını ve prensipte aynı işletim sistemine sahip herhangi bir makinede çalışmasını sağlar. Ceza hızdır. Genellikle donanıma ne kadar yaklaşırsanız, işler o kadar hızlı çalışır.

Bunun dışında tüm programlar “özgün”dür (eğer programlamada böyle bir şey varsa) ve grafik uygulamalarına sıkı sıkıya bağlıdır.

2. 3B Dünya Modelleme

Bilgisayarların en büyüleyici yönlerinden biri, gerçekçi modeller oluşturmak için kullanılabilme biçimleridir. Gerçekçi bilgisayar oyunlarının ve daha ciddi bir uçta simülatörlerin büyük çekiciliği, bilgisayar ekranının icat edilmiş bir evrene açılan bir pencere gibi görünmesinin sağlanabilmesinden kaynaklanır - Sanal Gerçeklik. Yaratılış sürecine olan benzerlikten etkilenen bazı ünlü bilim insanları, bilgi 'parçalarından' inşa edilen gerçek bir Evrene dayalı gerçeklik teorilerini bile ele alacak kadar ileri gittiler. Tüm bunların temel önemi ne olursa olsun, bilgisayarların insan ifadesi ve deneyimi için yeni bir boyut sunduğu gerçeği değişmez. Basitçe söylemek gerekirse, Doğa yasalarının uygulanabileceği veya uygulanmayabileceği alternatif gerçeklikler yaratma olanağı sağlarlar. Her türlü garip ve egzotik durum icat edilebilir ve araştırılabilir. Günlük yaşamda (ve rüyalarda) karşılaşılan nesneler ve durumlarla en kolay ilişki kuran insanlar için, bilgisayar ekranında görünenler tanıdık gelmelidir. Bu tür modeller oluşturmak için büyük çaba harcanmıştır. Gerçekliği doğru bir şekilde tasvir etmesi gereken bir simülatörde, vurgu Doğa yasalarına tam olarak uyan modellerdedir.

Bu bölümde gerçek nesnelerin çok hızlı ve makul derecede doğru bir resmini sağlayan bir modelleme yoluna bakacağız. Çoğunlukla, ancak tamamen değil, bu yapı taşları olarak çokgen yüzlere sahip çok yüzlü yapıları, yani sözde 'vektör' grafiklerini içerir. Küreler ve yüksek derecede simetriye sahip diğer nesneler de hızlı bir şekilde çizilebilir. Aslında, rekoru düzeltmek için, vektör grafikleri başlangıçta başka bir şey ifade ediyordu. Monitördeki noktaların, ekranın bir bölümünden diğerine hızla değiştirilebilen bir elektron ışınıyla doğrudan birleştirildiği bir görüntüleme moduna verilen bir isimdi. Bu, ekrana çok fazla bellek ayırmayı gerektirmiyordu ve çok hızlı 'tel çerçeve' resimleri veriyordu. Günümüzdeki monitörlerdeki ekranlar bu tekniği kullanmıyor. Bunun yerine, görüntü, görüntünün bir tarafından diğer tarafına yatay raster taramalarından oluşturulur. Buna raster tarama (veya tarama dönüştürme) grafikleri denir. Bir taslağın raster taramalarla doldurulabildiği hız, onu çok kullanışlı bir teknik haline getirir. Ancak vektör grafikleri adı, görüntüleme teknolojisini değil, grafik modelleme tekniğinin kendisini tanımlamak için yaygın olarak kullanılmaya başlandı. Buradaki “vektör” sıfatı aslında programlarda vektör geometrisinin yoğun olarak kullanılmasına işaret ediyor.

Bir diğer önemli teknik ise SPRITES'ların önemli bir rol oynadığı Blok Görüntü Transferi türü grafiklerdir. Amiga'da, bu tür işlemleri çok hızlı bir şekilde gerçekleştiren BLITTER (BLock Image TransfER) adlı bir donanım parçası bulunur. Bu tür grafiklerde, bellek blokları bir bütün olarak işlenir ve bu çok faydalıdır çünkü bir kez RAM'e yerleştirildikten sonra tarama dönüşümünün ikinci kez yapılması gerekmez. Bayt bloğu basitçe ekran alanına taşınır. Bu şekilde, özellikle sprite'larla çok akıllıca ve hızlı şeyler yapılabilir, ancak görüntünün parçaları arasındaki ilişkiler esasen bloğun başlangıçta RAM'e nasıl yerleştirildiğine göre belirlenir. Sprite grafikleri bu kitapta daha fazla ele alınmayacaktır.

Bununla birlikte, popüler bilgisayarların gelecek neslinin, tartışacağımız 'vektör' grafikleri de dahil olmak üzere tüm yaygın grafik işlevlerinin donanım uygulamasına sahip olması muhtemeldir. Yakında tüm grafik işlevlerinin çok hızlı donanım 'geometri motorları' tarafından yapılması çok olasıdır.

2.1. 3-D Modelleme

"Gerçek zamanlı" 3B modelleme çok hızlı olmak zorundadır. Bunun nedeni, insanların resmin titremesini, her 50 milisaniyede bir kereden daha yavaş değişirse fark edebilmeleridir. Gerçek zamanlı çalışmak için, izleyicinin klavye, joystick veya fare aracılığıyla yeni veri girebilmesi ve etkilerini hemen görebilmesi gerekir. Bu zaman ölçeğinde en kolay dönüştürülebilen ve çizilebilen katı 3B yapılar çokyüzlülerdir.

Çokyüzlüler, birkaç çok iyi nedenden ötürü çok iyi grafik yapı taşları veya 'ilkel'lerdir:

* tamamen köşeleriyle tanımlanırlar,
* yüzler düz kenarlı çokgenlerdir,
* herhangi bir dönüşümde sadece köşelerin yeniden hesaplanması gerekir,
* dönüştürülmüş bir çokgen aynı zamanda bir çokgendir
* Çokgenler, raster taramaları kullanılarak hızlı bir şekilde 'katı' görünecek şekilde doldurulabilir.

Bütün bunların anlamı aslında, yüksek simetriye sahip olmayan (daireler gibi) eğimli yüzeylerde çizim ve gölgelendirme yapmanın çok zor olduğu ve eğimli yüzeyi olmayan tek 3 boyutlu nesnelerin çokyüzlüler olduğudur.

Aslında bilgisayar grafikleri, temel yapı taşları olarak çokyüzlülerin kullanımında tekel sahibi değildir. Gerçek dünya bunları yaygın olarak kullanır; birçok ev, altı kenarlı çokyüzlüler olan tuğlalardan yapılır.

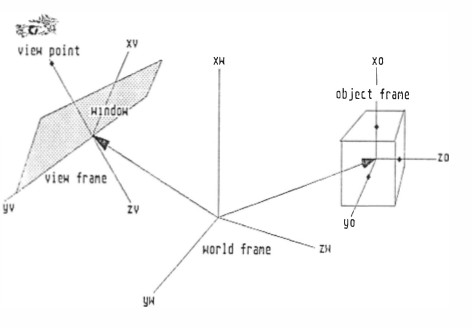
2.2. Dönüşümler ve Referans Çerçeveleri

Polihedrallerle ilgili yukarıdaki tüm ifadeler, doğrular ve düzlemler matematiğinin çok zarif ve kesin bir formülasyonu olan vektör cebiri adı verilen tanımlanmış bir matematiksel çerçeveye çevrilebilir. Matris biçiminde sunulduğunda daha da kullanışlı hale gelir ve bilgisayar grafikleri üzerine ders kitaplarında genellikle bu yaklaşım görülür. İleri matematik konusunda çok az bilgisi olan biri için bu çok korkutucu görünebilir. Aslında öyle değil. Birçok ortaokul müfredatı 2x2 matrisler kullanarak basit dönüşleri ele alır ve bundan çok daha karmaşık değildir. Matematik dünyasında yeni yollar açmak istemeyenlerimiz için bu, genel yöntemi anlamak ve sonuçları güvenle almak meselesidir. Sonuçta, dönüşümlerin çalıştığını gördüğünüzde bunları programlarınızda kullanabilir ve unutabilirsiniz. Tekerleği yeniden icat etmeye gerek yok.

Ancak şimdilik, sorunu bütünüyle ortaya koymak için, Şekil 2.1'de gösterildiği gibi, dönüşümlerin çeşitli aşamalarını ele alalım. Görünüm çerçevesi ile dünya çerçevesi arasındaki ayrım ve bunlar arasındaki dönüşümler, Ek 6'da daha ayrıntılı olarak tartışılmaktadır .

2.2.1. Nesne Çerçevesi

Bilgisayarda var olan bir nesnenin ekranda görünmeden önce oldukça karmaşık bir yaşamı vardır. Bu karmaşıklığın çoğu, onu 'gerçekçi' kılmak için gereken çeşitli dönüşümlerden kaynaklanır. Ancak bunlar ne olursa olsun (döndürmeler, çeviri veya daha egzotik bir şey), nesne orijinal kimliğini, yani göreli boyutlarını korumalıdır. Bunun anlamı, hiçbir hesaplamanın kesinlikle kesin olamayacağı ve her saniye 20 kereden daha hızlı yeniden hesaplanan resimle, orijinal tanıma sürekli olarak başvurulmazsa, birikimli hataların onu tanınmaz hale getirmesi uzun sürmez (bu sorun, hız için yalnızca sınırlı doğrulukta yapılan tüm hesaplamalarımızda ortaya çıkar). Bu nedenle, nesneyi tanımlayan orijinal verilere sürekli olarak geri dönmek gerekir. Nesnenin tanımlandığı bu yere nesne çerçevesi adını veriyoruz (bu isimde kutsal bir şey yoktur, başka insanlar başka isimler icat etmiştir). Elbette gerçek anlamda 'var' değildir, sadece köşelerin konumlarını sabitleyen sayılar bir orijinden ölçülen koordinatlardır. Bu orijinin nesne çerçevesinin bulunduğu yer olduğu söylenir. Nesne çerçevesi, nesnenin simetrisini yansıtacak şekilde konumlandırılabilir. Örneğin, bir küpün doğal nesne çerçevesi, küpün kenarları koordinat sisteminin x, y ve z eksenlerine paralel olacak şekilde, küpün simetri merkezinde (ağırlık merkezi) merkezlenmiş bir Kartezyen (x,y,z) koordinat sistemi olabilir (Şekil 2.1'de gösterildiği gibi).



*Şekil 2.1 Referans çerçeveleri*

Özellikle karmaşık bir nesne birkaç basit nesneden oluştuğunda, bir araya getirilmiş birkaç nesne çerçevesi olabilir. Basit nesneleri (ilkelleri) bir araya getirip karmaşık bir nesne oluşturma süreci, tam olarak bahsettiğimiz türden dönüşümleri içerir. Bu dönüşümlere bazen örnek dönüşümleri denir.

2.2.2. Dünya Çerçevesi

Karmaşık bir nesne inşa ettikten sonra — yaratmak üzere olduğumuz senaryoda bir 'aktör' olarak düşünülebilir — onu diğer tüm 'aktörler' ile birlikte arenaya yerleştirmek gerekir. Tüm nesnelerin yaşadığı bu ortak alana dünya çerçevesi denir. Doğa Yasalarının rol oynadığı yerdir. Örneğin, herhangi bir kuvvete maruz kalmayan nesneler ya hareketsiz kalır ya da sabit hızla hareket eder. Bu Newton'un Birinci Yasasıdır. Bu dünya bizim yaratımımız olduğundan, istersek bu yasalara bağlı kalmak zorunda değiliz. Burası çarpışmaların test edildiği yerdir. Nesneyi dünya çerçevesindeki son konumuna taşıyan dönüşüme nesneden dünyaya dönüşüm adını vereceğiz. Bu, dönme ve ötelemenin bir kombinasyonundan oluşacaktır.

2.2.3. Görünüm Çerçevesi

Gerçek dünyadaki herkesin ona dair farklı bir görüşü vardır ve aynı şey bilgisayarın içinde yarattığımız dünya için de geçerlidir. Tek fark, yalnızca bir ekranın ve dolayısıyla yalnızca bir izleyicinin olmasıdır. Dünyaya dair görüş, gözlemcinin nerede durduğuna ve baktığına bağlıdır.

Gözlemcinin gördüğü dünya görüşü en kolay şekilde görüş çerçevesi ile temsil edilir. Bu, gözlemcinin bakışını takip eden x, y ve z eksenlerinden oluşan bir kümedir. Genellikle z ekseni ileriye bakar ve bizim geleneğimizde x ekseni dikey olarak yukarı bakar. Bu resimde, 100 mesafede düz bir şekilde duran bir nesne görüş çerçevesinde (0,0,100) koordinatlarına sahip olacak ve gözlemci 90 derece sola dönerse görüş çerçevesi koordinatlarına (0,100,0) sahip olacaktır. Genel olarak görüş çerçevesinin dünya çerçevesindeki konumu sürekli olarak değişecektir. Örneğin bir uçuş simülatöründe görüş çerçevesi kokpitten görünen görüştür.

İlk bakışta tüm bu referans çerçevelerinde gereksiz bir bakış açısı tekrarı olduğu düşünülebilir. Ancak, gözlemcinin ve grafik ilkellerinin (nesnelerin) göreceli hareketlerini hesaba katmayı kolaylaştırmak için genel resmin oluşturulabileceği doğal bir hiyerarşi tanımlarlar.

Özellikle bir şeye dikkat etmek gerekir. Görüntü çerçevesini sola döndürmek veya sahneyi sağa hareket ettirmek aynı göreli hareketi verir ve ekranda aynı resmi verir. Bu, iki hareket arasında basit bir bağlantı olduğunu gösterir. Matematik dilinde, birinin diğerinin tersi olduğu söylenir. Dönmelere ayrıntılı olarak baktığımızda buna tekrar döneceğiz. Bu nokta Ek 6'da ayrıntılı olarak incelenmektedir .

2.2.4. Ekran

Bu mantıksal ekrandır, resimlerin görüntülenmeden önce çizildiği RAM bloğudur. 3. Bölüm'de açıklandığı gibi, RAM'in ekrana tahsis edilme şeklini izleyerek haritalanır ve bu da ekran çözünürlüğüne bağlıdır . Bu, başlangıç ​​noktasının (ekran koordinatları (0,0) olan nokta) ekranın sol üst köşesinde olmasına neden olur. Görünüm çerçevesinden ekrana geçmek için, görüntülemek istediğimiz nesnelerin görünüm düzlemi adı verilen bir düzleme bir 'izdüşüm' yapmalıyız. Buna perspektif dönüşümü denir ve uzaydaki sıralamayı korumalıdır, böylece daha uzaktaki nesneler daha küçük görünür. Bu, nesnelerden gözlemcinin gözünün konumu olan görünüm noktasına "ışınlar" izleyerek yapılır. Bu ışınların görünüm düzlemiyle kesişimi, ekranda görünecekleri ana hatları tanımlar.

Ekran koordinat sistemine dönüşüm neredeyse son aşamadır, ancak tam olarak değil; ekranın sınırları vardır. Resmin bazı kısımlarının ekran RAM'inin dışında kaldığı ortaya çıkabilir; ekrana ayrılmış bellek kısmı. Noktaların görünür ekranda görünmesini kısıtlamak için hiçbir girişimde bulunulmazsa program bunları ekran RAM'inin dışında çizmeye çalışacaktır; bu da sistem çökmesine yol açabilir. Bu nedenle, görüntülenecek hiçbir noktanın ekran RAM'inin dışında kalmayacağı kesin olmadığı sürece, görüntü düzleminde görünenin yalnızca bir kısmı ekrana ulaşacaktır. Bu "pencere"dir. Görünmeyen şey "kesilmelidir". Ekranda pencereyi tanımlayan taslağa görüntü portu denir. Son görüntüyü üretmek için harcanan çabayı açıkça ifade etmek için buna bazen kırpma çerçevesi de denir.

Hatta görüntü çerçevesinin kendisinde üç boyutu kırpmak bile gerekir. Gözlemciden çok uzakta olan nesneler görüntülenmemeli ve onlar hakkında endişelenerek zaman harcanmamalıdır. Ekranda sınırlı bir çizim çözünürlüğüne sahip olmanın bir sonucu olarak küçük nesneler kötü bir şekilde bozulur. Sonuç olarak tüm çok uzaktaki nesneler tek pikseller olarak son bulur ve ufukta her tarafta bir nokta kümesi olabilir. Paralel çizgi kümeleri sonunda tek bir çizgiye yakınsar ve bu çizginin yoğunluğu asla azalmaz. Tüm bunları durdurmak için görüntü çerçevesinin kökeninden belirli bir uzaklıktan daha uzakta olan nesneleri tamamen kırpmak mantıklıdır.

2.3. Koordinat Sistemleri

Tüm bu dönüşümleri matematiksel bir temele oturtmaya çalıştığımızda hemen zor ve sinir bozucu bir sorunla karşılaşırız: Koordinat sistemlerini nasıl tanımlayacağımız. Mühendislikte, bilimde ve matematiğin çoğunda sağ elli Kartezyen koordinatlarda çalışmak standarttır. Sağ elli ve sol elli Kartezyen koordinat sistemleri Şekil 2.2'de gösterilmiştir. Bu kurala uygun olarak her zaman sağ elli bir Kartezyen koordinat sistemi kullanacağız. Ancak uyaralım, bu bilgisayar grafikleri dünyasında standart değildir. Sol elli sistemler bol miktarda bulunur ve bazen her iki kural da aynı anda kullanılır!

Bilgisayar grafiklerinde sıkça kullanılan bir başka kural daha vardır ki, buna bağlı kalacaksak, eksenlerin görüş çerçevesindeki yönelimini zorlar. Pozitif z ekseninin gözlemcinin baktığı yönde, resme doğru ileriye doğru işaret etmesidir.

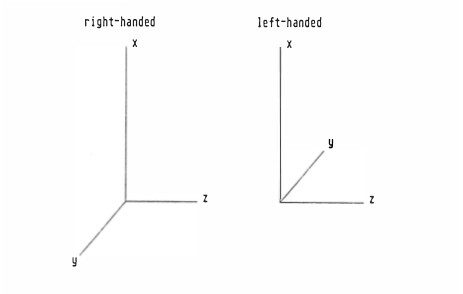
Tüm bunları bir araya getirerek, Şekil 2.1'de gösterilen çeşitli koordinat sistemlerine ulaşmayı seçtik. Pozitif x yukarıyı ifade eder ve dünya çerçevesinde yz düzlemi yer seviyesini tanımlar.

Koordinat sistemleri ve referans çerçeveleri de Ek 6'da ele alınmaktadır .

2.4. Vektörler ve Matrisler

Bilgisayarı seven ama matematiği sevmeyen biri için matris ve vektörlerin tanıtılması pek de hoş karşılanmaz. Tüm gerekli matematiği doğrudan cebirle yapmak mümkün olsa da vektörler ve matrisler, içinde çalışmak için zarif ve tutarlı bir çerçeve oluşturur. Ayrıca matrislerin onları özellikle yararlı kılan özellikleri vardır. Bir örnek, bir dizi dönüşümün ardışık olarak gerçekleştiği zamandır, örneğin bir nesnenin x ekseni etrafında dönmesinin ardından y ekseni etrafında dönmesi gibi. Nesnenin koordinatlarını her dönüşten sonra iki kez hesaplamak yerine, iki dönüşüm matrisini birleştirmek (birbirleriyle çarpmak) ve ardından birleşik dönüşümü yalnızca bir kez gerçekleştirmek mümkündür. Dönüştürülecek çok sayıda nokta olduğunda bu çok zaman kazandırabilir.

Çeşitli dönüşüm türlerini ortaya çıktıkça ayrıntılı olarak tartışacağız. Ek 5 ayrıca matrisleri ve vektörleri açıklar.



*Şekil 2.2 Sağ elli ve sol elli koordinat sistemleri*

2.4.1. Vektörler

Vektörler, belirli bir yöne nasıl gideceğinizi söyleyen matematiksel bir kısaltmadır. Vektörler matrislerle birlikte gider. Burada yine vektörlerle ilgili iki kural vardır. Vektörler satır vektörleri veya sütun vektörleri olabilir. Bu çok fazla bir şey ifade etmez, ancak bir vektörün yazıldığında nasıl göründüğünü ve dönüşüm matrisleri içindeki elemanların düzenini değiştirir. Mühendislik ve bilim öğretiminde vektörleri sütun biçiminde yazmak daha yaygındır ve kitap boyunca yalnızca bu kurala bağlı kalacağız.

2.5. Veri Yapıları

2.5.1. Değişkenler ve Etiketler

Assembly dilini ilk kez kullanırken alışılması en zor şeylerden biri cebirsel değişkenlerin olmaması, sadece kayıt defterlerinde ve bellek konumlarında saklanan verilerin olmasıdır. x'i y'ye ekleyemezsiniz ancak *d0 kaydının içeriğini d1* kaydının içeriğine ekleyebilirsiniz . Amiga gibi 16 bitlik bir sistemde bellek konumlarını bulmak bile zorlaşır çünkü program yazıldığında her zaman bilinmezler (İşletim Sistemi tarafından kullanılan ve sabit olan kayıt defterlerinin adresleri hariç). Bu, *PEEK* ve *POKE'nin* RAM'deki herhangi bir yere, sabit olacak ve program tarafından her zaman erişilebilir olacak adreslere erişim sağladığı daha basit 8 bitlik mikroişlemcilerle tezat oluşturur. Amiga gibi gelişmiş bir işletim sistemine sahip bir mikroişlemcideki sorun, bir program makineye gerçekten yüklenip çalışmaya hazır olana kadar tam konumunun bilinmeyecek olmasıdır. Mutlak kod ( *ORG* derleyici yönergesi tarafından belirlenen) kullanılarak İşletim Sisteminin programı belirli bir bellek konumunda yüklemesini zorlamanın bir yolu vardır ancak bu, programa esneklik kazandırır ve diğer yazılımlarla çakışmalara yol açabilir. Bu, bilgisayarı tamamen kendine bağlayacak bir oyun için sorun olmayabilir ancak işletim sistemindeki sonraki değişikliklerin kurbanı olabilir.

Genel felsefe, tüm bunlardan izole edilmiş ve RAM'in herhangi bir yerinde konumlandırılabilen ve çalıştırılabilen, tamamen kendi kendine yeten paketler olarak gelen programlar üretmektir. İlk bakışta bu yaklaşımla ilgili aşılmaz sorunlar var gibi görünüyor: Bir veri tablosu nasıl kurulur ve daha sonra nasıl bulunur ve bir testin sonucuna bağlı olarak yürütülecek alt rutinlerin adres tablosu (atlama vektörleri) nasıl kurulur? Bu sorunlara çeşitli çözümler vardır, bunlardan bazıları işlemcinin belirli adresleme modlarını kullanırken diğerleri, Bölüm 1'deki konumdan bağımsız ve yeniden konumlandırılabilir kod tartışmasında daha önce belirttiğimiz gibi, derleyiciye güvenir . Sorun, daha sonra hesaplanacak adreslerin geçici ikameleri olan etiketlerin kapsamlı kullanımıyla çözülür.

Etiketler herhangi bir derleyici programında çok önemli bir rol oynar. Kodda göründükleri şekilde cebirsel değişkenler gibi görünürler ancak öyle değildirler. Bir etiket, bir değişkenin geçerli değerinin tutulduğu bir bellek konumuna işaret eden bir işaretçidir veya programın başka bir bölümüne işaret eden bir işaretçidir. Zorluğun çoğu burada ortaya çıkar.

2.5.2. Listeler

Verileri verimli bir şekilde depolamanın ve erişmenin yollarını bulmak, bilişimde yoğun bir çalışmanın konusu olmuştur. Bilgisayar grafiklerinde, özellikle hızın önemli olduğu yerlerde çok önemlidir. Önemli olan, verileri eldeki sorun için kolayca erişilebilecek bir biçimde depolamaktır. Her zaman tüm uygulamalar için en iyi biçimde olmayabilir ve bu süreçte bazı manipülasyonlar gerekebilir.

İlkel öğelerin çokgensel yüzlere sahip çok yüzlü yapılar tarafından modellendiği vektör grafiklerinde en önemli olan, köşelerin (köşelerin) listeleri ve bunları birleştiren düz çizgi kenarlarıdır. Şekil 2.3, bu şekilde modellenmiş bir evi göstermektedir. Bu yapıyı tanımlamak için bir veri listesi oluşturmanın birden fazla yolu vardır, ancak en yaygın olarak kullanacağımız yolun merkezinde, yüzeyleri benzersiz bir şekilde tanımlayan bağlantıların listesi bulunur: kenar listesi. Kaçınılması gereken bir şey, köşelerin gerçek koordinatlarını birden fazla kez tekrarlamak zorunda kalmaktır. Her köşeye bir numara vermek ve bunun yerine buna başvurmak daha iyidir. Bir köşenin x, y ve z koordinatları gerektiğinde, listedeki konum basitçe köşe numarasıyla ilişkili olduğu sürece, ST'nin güçlü dizinli adresleme modları tarafından koordinat listesinden çizilebilirler. Bu noktayı açıklığa kavuşturmak için, evi çizmek için gereken listeler şunlardır. Her yüzeyin rengi vb. gibi diğer nitelikleri içeren başka listeler de olacaktır, ancak bunlar burada gösterilmemiştir. Ev çok karmaşık değil, ama gerçekten karmaşık bir nesne için listelerin ne kadar uzun olabileceğini göstermeye yetecek kadar karmaşık.

Öncelikle evin tamamındaki poligon sayısı belirtilmelidir. Her düzlemsel yüz uygundur: dört duvar, iki eğimli çatı, bir kat, bir kapı, yani:

yüzey numarası: 8

Burada yalnızca bir giriş var ancak başka binalar olsaydı bu bir liste olurdu. Daha sonra her yüzeydeki kenar sayısı verilir, burada giriş, şekilde gösterildiği gibi yüzeyin numarasıyla (çember içine alınmış) aynı konuma sahiptir:

kenar sayıları: 5, 4, 5, 4, 4, 4, 4, 4

Bundan sonra, dış yüzeyin etrafında saat yönünde giden tepe noktası numaralarının sıralı listesi kenar listesini oluşturur. Verileri program için en yararlı hale getirmek için, her yüzey için ilk tepe noktası, kapalı bir döngü oluşturmak için grubunun sonunda tekrarlanır.

kenar listesi: 7,8,9,2,1,7,1,2,3,4,1,4,3,10,5,6,4,6,5,8,7,6,5,10,9,8,5,2,9,10,3,2 1,4,6,7,1,11,12,13,14,11

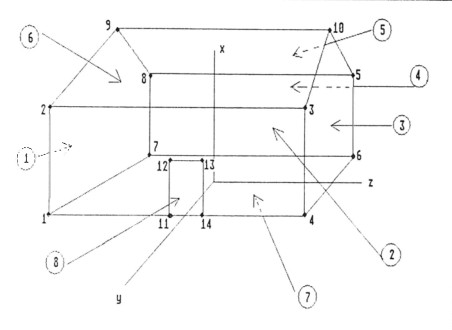
Son olarak, hangi ölçekte kullanılırsa kullanılsın, x, y ve z için tepe noktası numaraları sırasına göre gerçek koordinatlar verilir:

x koordinatları: 0,100,100,0,100,0,0,100,150,150,0,50,50,0

y koordinatları: 50,50,50,50,-50,-50,-50,-50,0,0,50,50,50,50

z koordinatları: -100,-100,100,100,100,100,-100,-100,-100,100,-10,-10,10,10

Bu veriler, nesne çerçevesindeki evi tanımlamak için kullanılacaktır. Dünya çerçevesine dönüşümün ardından bazı listeler, kenar listesi, kenar numaraları ve yüzey numarası değişmeden kalacaktır ancak dünya çerçevesindeki koordinatlar farklı olacaktır.



*Şekil 2.3 Çokgen ağ olarak modellenmiş bir ev*

2.6. Özet

3D grafiklerin bu matematiksel yönlerine karşı tutumunuz ne olmalıdır? Eğer matematiksel eğiliminiz varsa, o zaman neler olup bittiğini ayrıntılı olarak anlamaya çalışmak mantıklıdır. Bu size kendi dönüşümlerinizi yazma ve üretilebilecek çok ilginç efektlerden bazılarını keşfetme gücü verir. Eğer matematiksel eğiliminiz yoksa, matematiksel dönüşümleri gerektiği gibi "takılacak" yazılım "kara kutuları" olarak düşünün. Bu kitaptaki dönüşümler bunu yapmanıza izin verecek şekilde yapılandırılmıştır. Tek yapmanız gereken onlara verileri nasıl sunacağınızı anlamaktır.

3. Ekranda Çizim

Bu bölümde Amiga ekranının nasıl ele alındığına bakıyoruz. Bu, Amiga'ya özgü ancak hızlı grafikler için büyük önem taşıyan bir ayrıntıdır çünkü amacımız gerçek zamanlı olarak 3B katı nesneler çizmektir. Bunun çok önemli bir yönü, çokgen şekilleri hızlı bir şekilde doldurmak olacaktır.

Grafik programları ne kadar karmaşık olursa olsun, eninde sonunda çıktıları ekranda görünmelidir. Yeni programcılar için görünür çıktı üretmekle ilişkili bir dizi kafa karıştırıcı terim vardır: oyun alanları, bit düzlemleri, ekranlar. Bu kavramlar Amiga'ya yerleştirilmiş güç ve esneklik nedeniyle ortaya çıkar.

Basitçe söylemek gerekirse, Amiga 2 boyutlu sprite grafiklerini (Pacman grafikleri, ikonlar ve kaydırmalı türdeki birçok popüler oyun) uygulamak için güçlü bir araç setiyle tasarlanmıştır ve bu grafik terimlerine yansır. Bu, 3 boyutlu grafiklerin zor olduğu anlamına gelmez; aksine çok iyi karşılanmıştır ancak göreceğimiz gibi, 3 boyutlu grafikler için Amiga'da bulunan grafik cephaneliğinin yalnızca küçük bir kısmının kullanılması gerekir.

Öncelikle monitör ekranında görünen resmin RAM'in içeriğinin doğrudan bir "haritası" olduğunu unutmayın. Burada kullanıldığı şekliyle harita kelimesi biraz matematikçi jargonudur. RAM'de bulunan şeyin monitör ekranındaki şeyi tamamen tanımladığı anlamına gelir, ancak buna hiç benzemeyecektir. RAM, donanımın 2 boyutlu resme dönüştürdüğü bitişik (hepsi bir satırda) 1 boyutlu baytlardan oluşan bir bellek dizisidir. Hayatı kolaylaştırmak için, resimleri görüntülemeye ayrılmış bellek bölümüne özel bir ad vereceğiz -Video RAM- bilgisayar sistemlerinde yaygın olarak kullanılan bir terim. Amiga'da Video RAM, tüm özel donanımlar (Agnus, Paula ve Denise olarak adlandırılır) tarafından erişilebilen ve Chip Ram olarak adlandırılan bellek bölümünde yer almalıdır. Video RAM'in temel gereksinimleri, kolay çizime izin verecek ve renk bilgilerini tutacak şekilde düzenlenmesidir. Rengin nasıl dahil edildiğini anlamak, Video RAM'in düzenini anlamanın anahtarıdır.

Yapacağımız grafiklerde ek ve önemli bir komplikasyon daha var. Buna çift arabelleğe alma veya ekran arabelleğe alma denir ve titreşimsiz resimler için olmazsa olmazdır. Bunun anlamı, iki adet Video RAM parçasının kullanılabilir olmasıdır: biri üzerine çizim yapmak için diğeri ise görüntülemek için. Atari ST'de bunlara sırasıyla mantıksal ve fiziksel ekran adları verilmiştir. Bunlar ileri geri değiştirilir, böylece bir ekranda (mantıksal) bir resim çizilirken, son tamamlanmış resmi tutan diğeri (fiziksel) monitörde görüntülenir. Daha sonra yeni resim tamamlandığında görüntüye alınır ve fiziksel ekran haline gelir. Eski fiziksel ekran daha sonra yeni mantıksal ekran haline gelir ve bir sonraki resmi çizmeye hazır hale gelecek şekilde silinir. Her yeni resmi, film anlamında bir kare olarak düşünmek yardımcı olur, böylece gerçek zamanlı grafikler etkileşimli bir film gibi gelişir.

Programcı, bir ekrandan diğerine geçişin programla doğal olarak senkronize olmasını sağlar; program, yeni çerçeve tamamlanana kadar geçişi istemez ve donanım, ekrandaki raster sağ alt köşeye ulaşana ve tekrar yukarıya uçmaya hazır olana kadar görüntüyü değiştirmez. Bunun gerçekleşmesi için geçen kısa süre — dikey boşluk olarak adlandırılır — donanımın ekranları değiştirmesi için fazlasıyla yeterlidir.

3.1. Ekran

Sorunu anlamak için, gerçek monitör ekranı ile görüntüyü tutan Video RAM bloğu arasındaki farkları düşünün. Monitör ekranı, bir elektron ışınının üzerine yazdığı bir katot ışın tüpünün dikdörtgen ucudur. Bunu bir resim gibi göstermek için ışın, bir dizi 'raster' taramada soldan sağa ve yukarıdan aşağıya çok hızlı hareket eder; resim, her biri piksel adı verilen birimlerden oluşan yakın aralıklı yatay çizgilerden oluşur. Aslında hiç de sağlam bir resim yoktur, sadece uzaktan öyle görünür. Bunu kendiniz görmek için monitör ekranını bir büyüteçle yakından inceleyin.

Öte yandan bellek, mikroişlemcinin doğrudan adresleyebileceği en küçük elemanlar olan bitişik bir bayt satırı olarak düzenlenmiştir. Bunlardan en küçük çözülebilir birim bittir (8 bit = 1 bayt). Bir şekilde bellekteki her bit, ekrandaki en küçük 'nokta' veya pikselle doğrudan ilişkili olmalıdır. Amiga'nın büyük esnekliği, satır sayısına ve piksel boyutuna bağlı olarak birçok değişikliğe izin verir. Bu ikisi birlikte ekran çözünürlüğünü oluşturur: satır sayısı ve piksel ne kadar küçükse, çözünürlük o kadar yüksek olur. Bu kitaptaki programlarda gösterilen durumu inceleyeceğiz: 32 renk ve 320x200 piksel matrisi (320 eninde ve 200 yukarı) ile sözde düşük çözünürlük. Bu mod, iyi bir renk aralığına sahip hızlı gerçek zamanlı resimler üretmemizi sağlar. Diğer modlar daha yüksek çözünürlüklü ve daha renkli olabilir, ancak çok yavaştır. İlgilenen okuyucu, diğer modların ayrıntıları için Amiga Donanım Referans Kılavuzuna yönlendirilir.

3.1.1. Oyun Alanları ve Sprite'lar

Amiga grafik sisteminin kalbinde oyun alanı vardır. Oyun alanı aslında mantıksal ekranımızdan başka bir şey değildir. Oyun alanları, Pacman benzeri resimler ve simgeler için çok güçlü olan 2 boyutlu yönelimli sistemi oluşturmak için sprite'larla birlikte gelir. Sprite'lar, 16 bit (bir kelime) genişliğinde ve herhangi bir sayıda satır yüksekliğinde önceden çizilmiş resimlerdir ve oyun alanına çok hızlı bir şekilde kopyalanabilirler. Önce bellekte bir nesnenin farklı yönelimlerini gösteren birkaç sprite hazırlayarak ve sonra bunları oyun alanına art arda kopyalayarak, 3 boyutlu tip hareketi simüle etmek mümkündür. Bizim açımızdan bu bir hiledir. Oluşturacağımız grafik nesneler veya ilkel öğeler kendi başlarına bağımsız bir varlığa sahip olacak ve yalnızca çizim anında bir bayt desenine dönüştürüleceklerdir. Sprite'ları asla kullanmayacağımız için bunlar bizim için daha fazla ilgi çekici olmayacaktır.

Yani resmin çizileceği RAM çipindeki yer oyun alanı olarak adlandırılır. Amiga sistemine yerleştirilmiş esnekliğin bir sonucu olarak, oyun alanı monitör ekranı için çok büyük (veya çok küçük) olabilir. Bu bir felakete yol açmaz, sadece çok büyükse, oyun alanının yalnızca bir kısmının aynı anda görülebileceği anlamına gelir. Bu, arka planın ekran boyunca kaydırılabileceği oyunlar ve gösteriler için mükemmel bir düzenlemedir. Daha da karmaşık bir şekilde, tüm ekranı doldurmanız bile gerekmez. Resim, görüntüleme sırasında istenen boyuta "pencerelenebilir". Bir kez daha, bu varyasyonları kullanmayacağız. Bizim durumumuzda, oyun alanı monitörü tam olarak dolduracak ve pencere tüm oyun alanını kapsayacaktır. Bu, grafiklerimizi hiçbir şekilde sınırlamayacaktır.

Ekran çözünürlüğüne, boyuta ve renk sayısına karar verdikten sonra artık RAM'deki oyun alanı boyutunu hesaplayabiliriz. Bu da doğal olarak bit düzlemleri fikrine yol açar.

3.1.2. Bit-Düzlemleri ve Renk

Bir oyun alanı, monitörde gösterilecek resmi tutar ve bizim düşük çözünürlük durumumuzda, 200 satırı dikey, 320 pikseli yatay olarak görüntülemek ve 32 rengi göstermek için yeterli bilgiyi içermelidir. Renk büyük karmaşıklığı yaratır ve oyun alanının neden bu şekilde göründüğünü anlamanın anahtarıdır.

Neler olup bittiğini anlamak için önce en basit olasılığı, sadece 2 rengi ele alalım. Bu durumda her piksel açık veya kapalı olabilir. Bunu yapmak için oluşturabileceğimiz en basit oyun alanı, ilk 40 baytın (40 x 8 piksel) ilk raster tarama satırını, ikinci 40 baytın ikinci tarama satırını vb. temsil ettiği yerdir. Bu şekilde bellekteki her bit bir pikseli temsil eder; 1 olarak ayarlanırsa piksel açıktır ve 0 olarak ayarlanırsa piksel kapalıdır. Tüm ekranı doldurmak için oyun alanının 200 x 40 = 8000 bayt boyutunda olması gerekir. Bu durumda aslında bu şekilde yapılır. 8000 baytlık diziye bit düzlemi denir, çünkü tam olarak budur. Bu durumda oyun alanı tek bir bit düzlemi içerir. Bu şemada 4 renge kadar çıkmak bir sorun oluşturmaz, ancak şimdi 2 bit düzlemine ihtiyaç vardır. Ekrandaki her piksel ile her bit düzlemindeki belirli bir bit arasında hala birebir bir bağlantı vardır, ancak bitin 1 veya 0 olarak ayarlanması rengi sabitler. Örneğin, ekranın en üst sırasındaki ikinci pikselin koordinatları x = 1, y = 0'dır (x = 0, y = 0'ın (başlangıç ​​noktası) ekranın sol üst köşesinden başlaması bilgisayar ekranlarının tuhaf bir özelliğidir). Bu piksel, yalnızca iki adet 8000 bayt boyutunda RAM bloğu olan hem birinci hem de ikinci bit düzlemindeki ikinci bite karşılık gelir. Şimdi rengin kodlanabileceği yol açıklığa kavuşur. Her iki düzlemdeki bit 0 ise pikselin rengi 0'dır, düzlem 2'deki bit 0 ancak düzlem 1'deki bit 1 ise pikselin rengi 1'dir ve aşağıda gösterildiği gibi devam eder:

| **bit düzlemi numarası** | **1** | **2** | **renk** |
| --- | --- | --- | --- |
| bit değeri | 0 | 0 | 0 |
|  | 1 | 0 | 1 |
|  | 0 | 1 | 2 |
|  | 1 | 1 | 3 |

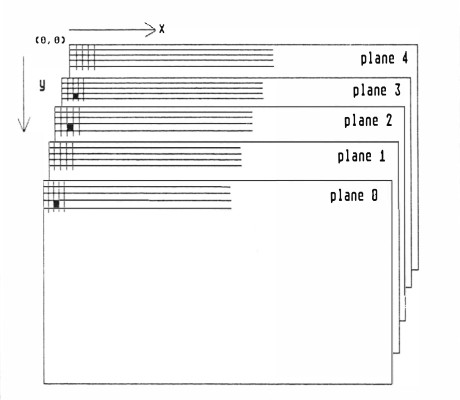
Bu nedenle, her bit düzlemindeki ikinci bitin nasıl ayarlandığına bağlı olarak ikinci piksel için 4 farklı renk olması mümkündür. Bunun için açıkça bir kalıp vardır: 1 bit düzlemi 2 renk, 2 bit düzlemi 4 renk, 3 bit düzlemi 8 renk, 4 bit düzlemi 16 renk ve 5 bit düzlemi 32 renk verir. Matematiksel olarak ifade edildiğinde, formül şudur:

renk sayısı = (2)\*\*bit düzlemlerinin sayısı.

Amiga'da bit düzlemleri ayrı 8000 bayt RAM bloklarıdır. Atari ST'de 4 bit düzlemi vardır ancak bunlar iç içe geçmiştir, böylece belirli bir pikselin ne yaptığını görmek o kadar kolay değildir.

Bizim amaçlarımız için 5 bit düzlemimiz ve dolayısıyla emrimizde 32 rengimiz var. Şekil 3.1 pikselleri (x=2,y=3) 13 rengiyle doldurmak için ayarlanmış 5 bit düzlemini göstermektedir.

Son bir soru daha var. Amiga 500'ün emrinde 4096 renk var; bu geniş aralık, 5 bitlik düzlemlerle görüntüleyebildiğimiz 32 renkle nasıl ilişkili? Cevap, herhangi bir anda toplam 4096 aralığından yalnızca 32'sini görüntüleyebileceğimizdir. Şu anda görüntülemek üzere seçtiğimiz renklerin (0 ila 31 arasındaki renkler) bu listesi, birlikte *Renk Tablosu* veya *Renk Paleti* adı verilen özel bir dizi kayıtta bulunur . Seçilen renklerin değerlerini başlangıçta tabloya yazmak ve herhangi bir zamanda değiştirilirlerse tabloyu yeniden yazmak programcının sorumluluğundadır. Amiga açıldığında nasıl kurulduğu olan standart paletin kodları *data\_00.s* dosyasında listelenmiştir . Özel donanım, renk tablosundaki 32 sayıyı ekrandaki renklere dönüştürmenin pratik ayrıntılarıyla ilgilenir. Elbette programcı tabloyu yazabilmek için hangi sayının hangi rengi oluşturduğunu bilmelidir; Bu noktayı daha sonra örnek programı incelerken ve yine 7. Bölümde daha detaylı olarak ele alacağız .



*Şekil 3.1: 32 renk için beş bit düzlemi*

3.1.3. Bakır Listeleri

Copper, grafik sisteminin neredeyse tamamını idare eden grafik yardımcı işlemcisi için kullanılan dostça bir isimdir. 68000 işlemcisinden bağımsız olarak çalışarak program yürütmeye devam etmesini sağlar. Tekrar ediyorum, buradaki amacımız Copper'ın farklı çözünürlük ve renk derinliğine sahip yatay dilimler içeren bir ekran ayarlamak gibi harika ve egzotik işlevlerinin tadını çıkarmak değil. Sadece örnek programlarda kullanılan Copper'ın işlevlerinin yönlerini tartışacağız.

Copper'ın bizim amaçlarımız için olmazsa olmaz bir işlevi vardır; oyun alanını monitör ekranına eşlemek. Çift arabellekli durumumuzda (çizim yapılacak bir ekran ve görüntülenecek başka bir ekran) dikey boşluk sırasında ekranları değiştirme gibi ek bir görev vardır. Copper'a tüm bunlarla ilgilenmesi için doğru zamanda doğru bilgi verilmelidir. Bu bilgiler Copper listelerinde toplanır.

Copper'ın işini yapması için gereken tek şey, iki ekranı oluşturan iki 5 bit düzleminin RAM'deki adresleridir. Bu adresler, her seferinde bir tane olmak üzere, Copper'ın ekranı oluşturmak için kullanacağı iki listeye karşılık gelen özel Sistem adresleriyle (şaşırtıcı olmayan bir şekilde Bit Düzlemi İşaretçileri olarak adlandırılır) birlikte girilir. Elbette Copper'a kaç bit düzlemi kullanacağı da söylenecektir, bu yüzden Copper listemiz aslında her bit düzlemi adresini hedef Sistem kaydına bağlayan bir dizi bağlantıdan başka bir şey değildir. Ekranı oluşturmak için Copper, dikey boşlukta, görüntülenecek bit düzlemlerinin adreslerini bit düzlemi işaretçilerine taşıyacaktır. Donanım gerisini halledecektir. Daha fazla ayrıntı örneklerle verilmiştir.

3.2. Blitter ve Ekran

Agnus çipinde bulunan bir diğer donanım yardımcı işlemcisi olan blitter gerçekten de güçlü bir cihazdır. Varlığı Amiga'nın tüm işlemlerini gölgede bırakarak çok yüksek hızlı grafikleri mümkün kılar. BLIT kelimesi BLock Image Transfer'dan gelir ve kelimenin sesi, yapılabilecek güçlü işlemleri ima etmek için tam yerindedir.

Bizim amaçlarımız için blitter, bir oyun alanında bir resmi birleştirmenin birkaç temel aşamasında önemli bir rol oynar. Ana hatları çizmek, bunları doldurmak, bit düzlemlerine kopyalamak ve hızlı silme yapmak için kullanılabilir (ancak amaçlarımız için daha sonra açıklanacak nedenlerden dolayı çizgi çizmeyi atlayacağız). Her bakımdan büyük bir hız avantajı sağlar, 68000'den bağımsız temel grafik işlevleri gerçekleştirir ve programla devam etmesi için onu özgür bırakır. Blitter olmayan bir sistemde tüm grafik işlevleri ana işlemci tarafından yapılmak zorunda kalırdı, bu da işleri yavaşlatır.

Blitter'ın ana işlevi, baytları bir yerden diğerine çok hızlı bir şekilde kopyalamaktır. Bu, özellikle sprite'ların bir oyun alanı konumundan diğerine kopyalanarak "hareket ettiği" 2 boyutlu stil grafiklerde önemlidir. Bir sprite'ın dikdörtgen biçiminde kelime genişliğinde bir blok olduğunu unutmayın (bir kelime = 2 bayt); blitter yalnızca kelimelerden oluşan dikdörtgenleri işler. Bizim durumumuzda, ana hatların doldurulmasında ve görsellerin aktarılmasında kullanılacaktır. Bir resmin nasıl oluşturulacağına bir bakalım. Bir kez daha, bu blitter'ın işlemlerinin kapsamlı veya genel bir açıklaması değildir. Blitter'ın 3 boyutlu grafik "boru hattımızda" nasıl kullanıldığını açıklamaya yöneliktir.

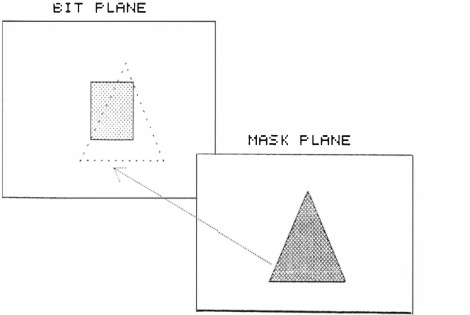
3.2.1. Bit Planes Düzeni — Genel Bakış

Örnek programlarda kullanılan genel bit düzlemi stratejisi bu aşamada açıklanırsa, neyin takip edileceğini anlamak daha kolay olacaktır. Yazar, bu stratejinin en iyisi olduğunu iddia etmemektedir. Diğer programcıların, şüphesiz, daha iyi fikirleri olacaktır. Ancak kullanılan yöntem, özel donanımın doğru, belgelenmiş kullanımına dayanır ve en derin düzeyde işleyişine dair mükemmel bir içgörü sağlar.

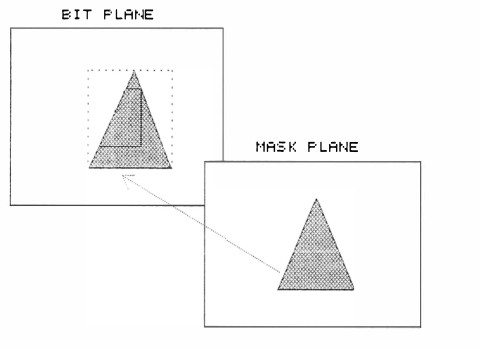
Mantıksal ekranda yeni bir resmin çizimi birkaç aşamayı takip eder. Bileşik resme eklenecek her yeni grafik nesnesi (ilkel), oyun alanı bit düzlemlerinden biri olmayan maske düzlemi adı verilen özel bir bit düzlemindeki dolu çokgenlerden çizilir. "Maske" adı, oyun alanı bit düzlemlerine halihazırda orada bulunanlar ve nesnenin rengi tarafından belirlenen bir şekilde yansıtıldığı için özel bir öneme sahiptir. Bunların hepsi oldukça karmaşık geliyor ve öyle de.

Sorunu açıkça anlamak için, blitter'ı her yerde kullandığımızı ve kurallara uymamız gerektiğini unutmayın. Blitter, RAM'in kelime çapındaki dikdörtgenlerinde çalışmayı sever ve bu yüzden görüntüyü tamamen çevreleyen bir dikdörtgen kopyalarız. Şimdi dikkatli olmalıyız çünkü maske düzlemindeki görüntüyü oyun alanına kopyalarsak, nesnenin yan tarafındaki her şeyi ve altındaki her şeyi içeren bütün bir dikdörtgeni karartacaktır. Yapmak istediğimiz şey, yalnızca nesnenin kendisi tarafından gizlenenleri örtmektir, blit dikdörtgenindeki her şeyi değil. Bir şekilde "arka planın" kendisi kopyalama sürecinde rol oynayacaktır.

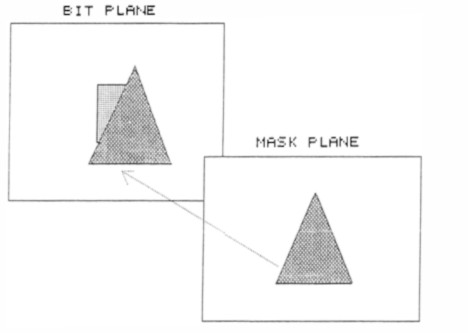
İşleri kolaylaştırmak için, yalnızca bir bit düzleminin olduğu durumu ele alalım, ancak elbette her zaman beş tane olacak. Daha sonra birkaç bit düzleminin ek karmaşıklığını tartışacağız. Oyun alanında bir bit düzlemi ile, bir üçgen ve bir kareden bileşik bir resim oluşturduğumuzu varsayalım. Kare, oyun alanı bit düzlemine kopyalandı ve üçgen maske düzlemine çizildi ve bit düzlemine kopyalanmaya hazır. Şekil 3.2 neler olduğunu gösteriyor.



*Şekil 3.2: Üçgenin bit düzlemine kopyalanması*



*Şekil 3.3: İstemediğimiz şey - kare silindi*



*Şekil 3.4: Üçgenin doğru şekilde kopyalanması*

Üçgeni içeren dikdörtgen maskeyi bit düzlemine basitçe blitlersek, Şekil 3.3'te gösterildiği gibi tüm kareyi bloke ederdi. Bunun yerine istediğimiz şey Şekil 3.4'teki sonuçtur.

Blitter bunu halledebilir: ona sadece hedef arka plan dikdörtgeninin gösterilmesi gerekir ve "kurabiye kesme" işlevi (şekilli bir kesiciyle kurabiyeleri kesmek gibi) adı verilen akıllıca bir mantıkla, hiçbir şeyi gizlemeden hem üçgeni hem de kareyi birleştirir.

32 rengi kodlamak için gereken 5 bit düzleminin ek bir komplikasyonu daha var, onu tartışalım. Şimdi üçgenin bir renge sahip olduğunu düşünebiliriz — örneğin renk 5. Bunun ne renk olduğunu henüz bilmiyoruz; renk kaydı 5'te hangi sayıyı ayarlamayı seçtiğimize bağlı olarak herhangi bir şey olabilir. Renk 5'in anlamı, üçgenin içinde bitlerin bit düzlemleri 1 ve 3'te 1 olarak ayarlanması ve diğer tüm düzlemlerde 0 olarak temizlenmesi gerektiğidir. Sadece düzlemler 1 ve 3'teki bitleri ayarlamak ve diğerlerini yok saymak yeterli değildir. Dolayısıyla nesnenin bir maske olarak gerçek anlamı ortaya çıkar; maske, bir şablon gibi, renge bağlı olarak bazı bit düzlemlerinde bitleri 1 olarak ayarlamalı ve diğerlerinde bitleri 0 olarak temizlemelidir.

Yukarıdaki tartışmanın tamamı uzun programlama ve sıkı çalışma gibi geliyor. Öyle değil. Bu karmaşık manevralar, blitter kayıtlarına uygun kodları yazarak kolayca gerçekleştirilebilir. Hangi kodların yazılacağını anlamak başka bir şeydir. Ancak örnek program sunulana kadar ayrıntıları geciktireceğiz, böylece bakabileceğiniz bir şeyiniz olur.

Son olarak blitter, bir sonraki kare çizilmeden hemen önce mantıksal ekranı silmek için kullanılabilir. Beş bit düzlemini (40.000 bayt) bir defada silmenin, blitter tarafından muazzam şekilde yardımcı olan zaman alıcı bir işlem olduğu ortaya çıktı.

3.3. Çizim

Hızlı grafik programımızın kalbinde çokgenleri çizen ve dolduran rutinler yer alır. Çokyüzlüleri katı 3B nesneler için model olarak kullanmak, doldurulacak çok sayıda çokgen yüzey üretecektir. İş en iyi iki aşamada yapılır: önce köşelerdeki köşeleri birleştirerek bir taslak çizilir, sonra taslak içindeki bölge doldurulur. Bunun kulağa geldiği kadar basit olmadığı ortaya çıkar. Bunun nedenini anlamak için, çizgi çizmenin ve bölge doldurmanın ne kadar hızlı yapılabileceğine bakalım.

Bilgisayar ekranında hızlı çizgi çizme prosedürü uzun zamandır var. Bresenham algoritması olarak adlandırılıyor. Hızlıdır çünkü aritmetik kullanmaz — özellikle 68000 setindeki en yavaş talimatlar arasında yer alan bölme veya çarpmayı kullanmaz. Blitter'ın kendisi çizgiler çizme ve hatta blitter doldurulacak ana hatlar için özel çizgiler çizme yeteneğine sahiptir, ancak rutinlerimiz çizgi çizmek için blitter'ı kullanmaz. Bunun nedeni nedir? Cevap, blitter'ın ana hatları çizme biçiminde ve bunların doldurulması için nasıl görünmesini beklediğinde yatmaktadır.

Blitter, yalnızca ekranın en üstünden başlayıp her seferinde bir satır aşağı doğru ilerleyen bir dizi raster taramasıyla kapalı bir poligonu doldurabilir. Her satırda yalnızca iki piksel kümesi bulmayı bekler ve ilk pikselden doldurmaya başlar ve ikinci pikselde durur. Şimdi sorunu görüyorsunuz. Tarama satırı başına ikiden fazla piksel kümesi bulursa başı dertte demektir. Diyelim ki, kazara üç piksel kümesi var. İlk pikselden doldurmaya başlayacak, ikinci pikselde doldurmayı bırakacak ve üçüncü pikselden tekrar başlayacak. Ancak duracağı dördüncü bir piksel olmadığında ekranın kenarına kadar dolmaya devam edecektir ki bu istenen bir şey değildir.

Dikkatle seçilmiş çoğu durum için, sorun çıkmaması için blitter'ın özel bir çizgi çizme modu seçilebilir ancak 3 boyutlu grafiklerin dinamik hareketiyle, bazen blitter'ın tarama çizgisi başına iki pikselden fazla ayarlanmış bir anahat çizdiği durumlar ortaya çıkar. Bu aslında blitter'ın bir hatası değil, anahatta bağlanacak çokgen köşelerinin konumlarına ilişkin bilginin ona iletilmesinin bir komplikasyonudur. Sorunu genel durumda çözmek zahmetli hale gelir, bu nedenle bu kitaptaki rutinler için blitter yerine her zaman işe yarayan özel rutinler kullanılarak sorun önlenmiştir. Anahat çizmek diğer tüm işlere kıyasla çok hızlı olduğundan burada hız kaybı olmaz.

3.3.1. Bresenham Çizgi Çizim Algoritması

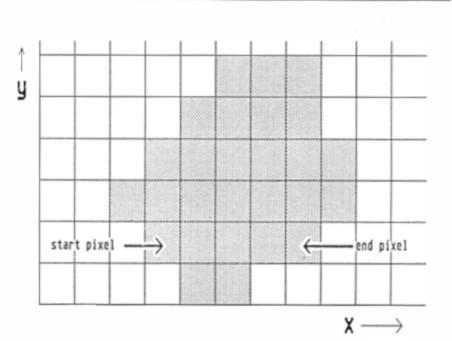
Denklemi kullanmadan iki nokta arasında nasıl çizgi çizebilirsiniz?

y = mx + c

Verilen bir x değeri için y değerini elde etmek için bir çarpmanın gerekli olduğu durum nedir?

Neyse ki bu sorunun çözümü yıllar önce 1962'de JE Bresenham tarafından çözüldü. O zamanki sorun, ne çarpabilen ne de bölebilen bir dijital çiziciyi kontrol etmekti. Bu tür işlemler 68000'de mevcuttur ancak zaman alıcıdır ve mümkün olduğunca bunlardan kaçınmak istiyoruz. Bresenham algoritmasının en büyük avantajı, yalnızca toplama ve çıkarmalar kullanarak bir çizginin tüm ekran koordinatlarını bulabilmesidir. Cebirsel terimlerle açıklandığında Bresenham algoritması korkutucu görünür ancak tüm harika fikirler gibi aslında çok basittir. Elbette bazı (hepsi olmasa da) ticari programlar, Bresenham yönteminin izin verdiğinden daha hızlı çizgiler çizen ve çokgenleri dolduran algoritmalar kullanır ancak anladıktan sonra daha iyisini yapmayı deneyebilirsiniz. Her durumda çok zarif ve çok hızlıdır.

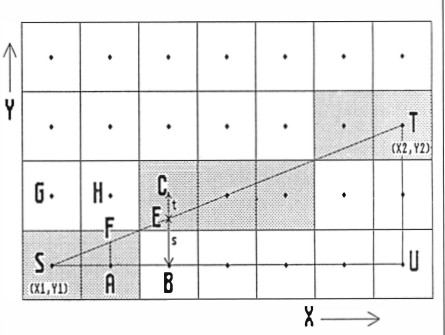
Karşı karşıya olduğumuz sorun, yatay çizgilerin dolgusunu yapmak için başlangıç ​​ve bitiş noktaları olarak kullanabilmemiz için bir çokgenin kenarları boyunca (x,y) koordinat çiftlerini bulmaktır. Piksellerin neden olduğu düzensizliği abartmak için seçilen çok küçük bir alanın dolgusu, Şekil 3.5'te gösterilmiştir. Sınırı bir çizgi olarak ele aldığımızda, farklı ekran çözünürlüklerinde farklı göründüğünü görüyoruz. Normal çözünürlükte, ekrandaki bir pikselin konumu yatay olarak 0 ile 319 arasında ve dikey olarak 0 ile 199 arasında bir tam sayı değeriyle belirtilir. Bu sınırlamayla, herhangi bir çizgi (yatay veya dikey olmadığı sürece) bir büyüteç altında bir merdiven gibi görünecektir. Bu, Şekil 3.6'da gösterilmiştir. Bir noktanın koordinatlarını ekran çözünürlüğünün izin verdiğinden daha iyi bir doğrulukla hesaplamaya çalışmamıza açıkça gerek yoktur, bu da tam sayı aritmetiğinin oldukça yeterli olduğu anlamına gelir. Ekrandaki bir noktanın konumunu 4 ondalık basamağa kadar hesaplamanın bir anlamı yoktur çünkü yalnızca hiçbir ondalık basamağa kadar çizilebilir. Bresenham stratejisi başarısını, ekranın piksel düzenine uyma şekline borçludur. İşte çalışma şekli.



*Şekil 3.5 Pikselleri göstermek için büyütülmüş küçük bir çokgen*

Ekranda S(x1,y1) noktasından başlayıp Şekil 3.6'da gösterildiği gibi T(x2,y2) noktasında biten bir çizgi çizdiğimizi varsayalım. Bu noktalar elbette tam olarak çizgi üzerinde yer alacaktır. Şimdi bir kalem ve cetvel alıp iki uç nokta arasında ideal bir matematiksel çizgi çizebilir ve ardından çizgiye en yakın olan pikselleri gölgelendirebiliriz. Çizgimiz ekranda bu şekilde görünecektir. Sonuç, piksellerin karelerle temsil edildiği şekilde gösterilmiştir. Bir algoritmanın, insan beyninin hangi noktaları gölgelendireceğine karar verirken otomatik olarak yaptığı şeyi yapmasını istiyoruz.

İşte bunu yapan Bresenham algoritması. Resmi daha basit hale getirmek için her pikseli merkezindeki bir noktayla değiştiriyoruz; bu, her pikselin ideal çizgiyi ne kadar ıskaladığını çok açık bir şekilde ortaya koyuyor. Diyelim ki tam olarak çizgi üzerinde olmayan A noktasına ulaştık ve bir sonraki adımın ne olacağını seçmek zorundayız. Bir sonraki nokta B(x+1,y) veya C(x+1,y+1) olabilir. Açık bir seçim gibi görünüyor; C noktası çünkü daha yakın. Bu anlamda daha yakın olmak, pikselin merkezinden E noktasında çizgiye olan dikey mesafenin daha kısa olması anlamına gelir. Buna hata diyebiliriz. Diyagramda, hata t, hata s'den küçüktür. Bu kararda bir şekilde H noktasını dikkate almadığımızı fark edin. Bunun nedeni, çizginin açısının 45°'den az olmasıdır. Açı 45°'den büyük olsaydı, H ve C noktalarını dikkate alırdık. Eğimi 1'den küçük olan çizgilerin (açı 45°'den küçük) eğimi 1'den büyük olan çizgilerden (açı 45°'den büyük) farklı bir durum olduğu zaten açıktır. Buna daha sonra geri döneceğiz.



*Şekil 3.6 Bir çizgi boyunca piksel konumları*

Pekala, sorun çözülmüş gibi görünüyor! Sadece B ve C gibi bir sonraki iki noktayı inceleyin, her birinin çizgiye olan dikey mesafesini hesaplayın ve daha kısa olanı seçin. Prensipte bu kadar. İdeal çizgiye kadar olan dikey mesafe pozitif bir hata (s gibi) ve çizgiye kadar olan dikey mesafe negatif bir hata (t gibi) olarak alınırsa, seçimin dayandığı genel nicelik (st) olur:

eğer (st) = D pozitif ise, bir sonraki nokta C'dir

Eğer (st) = D negatif ise bir sonraki nokta B'dir.

(st) niceliğine, açık sebeplerden dolayı karar değişkeni D adı verilir.

Bresenham'ın büyük yeniliği, bunu basit bir işlem haline getirmek için iki numarayı fark etmekti. Birincisi, yalnızca (st) işareti önemli olduğundan, (st) ile orantılı herhangi bir miktar işe yarayacaktır. İkincisi, bu hesaplamayı her seferinde tekrar yapmaya gerek olmamasıdır. Mevcut seçim için kullanılan D değeri, bir sonraki seçim için D değerini bulmak üzere hızla düzeltilebilir.

Yani şöyle oluyor. Güncellenen karar değişkeni D, pozitif mi negatif mi olduğunu görmek için test edilir. Negatifse ayarlanacak bir sonraki nokta B'dir. Sonra D buna göre güncellenir. Pozitifse ayarlanacak bir sonraki nokta C'dir. Sonra D buna göre güncellenir. Tek yapmamız gereken bu güncellemelerin ne olduğunu ve satırın en başında D'nin değerinin ne olması gerektiğini bulmaktır.

Bu soruları cevaplamanın anahtarı A'dan B'ye veya A'dan C'ye nasıl gidileceğine bakmaktır. A'dan B'ye gitmek için yatay bir hareket yapın; A'dan C'ye gitmek için önce yatay bir hareket yapın ve ardından dikey bir hareket yapın. Bireysel yatay ve dikey hareketlerle ilişkili hataları hesaplamak için S noktasına bakmak daha basittir. Buradan yatay bir hareket AF hatası üretir, ancak G'ye basit bir dikey hareket -SG hatası üretir (ideal çizginin altındaki noktalar pozitif hataya ve üstündeki noktalar negatif hataya sahiptir). Ancak SG, SA'ya eşittir, bu nedenle aslında sadece üçgen SAF'nin dikey ve yatay kenarlarının göreceli uzunluklarını dikkate almamız gerekir. Ancak çok önemli olan, üçgen SAF, genel üçgen SUT'a benzer ve kenarlar orantılıdır:

AF/SA = TU/SU = (y2-y1)/(x2-x1) = dy/dx

Burada dy, y'deki toplam mesafe ve dx, çizginin başlangıcından sonuna kadar olan x'deki toplam mesafedir.

Söylediğimiz gibi, orantılı olan her şey işe yarayacaktır, bu yüzden hatalar dy ve -dx olarak alınabilir. Her şeyi hala orantılı tutan 2'lik bir faktör daha bizi Bresenham'ın orijinal şemasıyla aynı çizgiye getirecektir:

basit yatay hareket: error = 2dy

basit dikey hareket: hata = -2dx

A'dan B'ye veya A'dan C'ye gerçek hareketler için:

yatay hareket (AB): error1 = 2dy

yatay artı dikey hareket (AC): error2 = 2dy-2dx

Bunlar bir sonraki seçim için karar değişkeni D'de yapılması gereken güncellemelerdir.

Son olarak, D'nin hangi değeriyle başlamalıyız? Başlangıç ​​değeri D1'i error1 ve error2'nin ortalama hatası olarak alırsak her şey yolunda gider.

D1 = (hata1 + hata2)/2 = 2dy-dx

Özetlemek gerekirse, algoritma şu şekildedir:

1. ilk noktayı x1,y1'e ve D'nin başlangıç ​​değerini D1'e başlat,
2. eğer D1 -ve ise, x'i artır ama y'yi artırma ve D = D + error1 yap, eğer D +ve ise, hem x'i hem de y'yi artır ve D = D + error2 yap
3. x = x2 olana kadar 2. adımı tekrarlayın.

Peki eğimi 1'den büyük olan doğrular ne olacak? Çözüm çok basit. Bunu net bir şekilde görmek için, bir parça izleme kağıdına eğimi 1'den büyük bir doğru çizin ve x ve y eksenlerini açıkça etiketleyin. Şimdi izleme kağıdını ters çevirin. Y ekseni yatay ve x ekseni dikey olduğunda, şimdi eğimi 1'den küçük olan orijinal doğrumuz gibi görünüyor, sadece x ve y eksenleri yer değiştirmiş durumda. Bu nedenle, formüllerde x ve y yer değiştirdiğinde her şey daha önce olduğu gibi çalışır.

3.3.2. Bresenham'ı Poligon Dolgusuna Göre Uyarlama

Blitter, ana hatları doldurmak için idealdir ancak dediğimiz gibi, bu ana hatların özel bir şekilde çizilmesi gerekir. Özellikle her tarama satırında yalnızca iki piksel kümesi bulmak ister: biri doldurmaya başlamak için ve diğeri durdurmak için.

Açıkladığımız prosedür kesinlikle bir doğru boyunca noktalar üretecektir, ancak bizim amacımız için hepsine ihtiyacımız yok. Eğimi 1'den küçük olan doğruları ele aldığımızda, "merdiven"in yatay kısmında bulunan S ve A gibi noktaların hepsinin y koordinatları aynıdır ancak x koordinatları farklıdır. Sadece ilkinin x koordinatı olan S gereklidir, çünkü A gibi diğerleri blitter'ı karıştıracaktır. Doğrudaki ilki, D işaretindeki değişikliği hemen takip eder. Bresenham algoritmasının bizim versiyonumuz, yalnızca dışbükey bir çokgeni doldurmak için raster taramaları için yatay doğruların başlangıç ​​ve bitiş koordinatlarını üretecek şekilde değiştirilmiştir. Bu, tek başına çizildiğinde yataylar boyunca deliklerle dolu bir doğru üreteceğinden, tam olarak normal anlamda bir Bresenham algoritması değildir. Ancak blitter bunu sever.

3.4. Örnek Program

Bu bölümde bir örnek program bulunmaktadır ancak oldukça uzundur ve yukarıdaki tartışmada yer alan tüm noktaları ve bunun dışında çok daha fazlasını göstermektedir. Çok fazla bir şey yapmaz; çift tamponlama ile birbiri ardına iki dolu üçgen çizer ancak kitabın sonraki aşamalarında ihtiyaç duyacağımız tüm rutinleri içerir. Üçgenlerin koordinatları *polydraw.s* dosyasının sonunda verilmiştir .

Burada rutinlerin mümkün olan en iyi veya en hızlı olduğu iddia edilmiyor. Diğer sürümler daha zarif ve daha hızlı olabilir. Ancak bu programlar hızlıdır ve işi yeterince yapar. Ayrıca, montaj kodu programlamanın çeşitli yönlerini ve Amiga donanımının işlevlerini göstererek eğitimsel bir değere sahiptirler. Nasıl çalıştıklarını incelediğinizde kendi geliştirmelerinizi yapmak isteyebilirsiniz.

Programların düzensiz bir karmaşaya dönüşmesini önlemek için, benzer türden alt rutinler içeren birkaç dosya ayarlanmıştır. Her bölümde farklı bir adı olacak bir ana kontrol programı her zaman olacaktır. Ek olarak, muhtemelen tüm önemli alt rutinleri içeren bir *çekirdek* dosya, "housekeeping" (ekranları silmek gibi) alt rutinleri içeren bir *systm* dosyası, değişkenlerin etiketlerini içeren bir *bss dosyası ve sayılar içeren bir veri* dosyası olacaktır. Dosyalar genellikle kaynak dosyaları olduklarını göstermek için *.s* uzantısıyla biter. Genellikle bu dosyalar , bir kez yazıldıktan sonra gelecekte de orada olmaları için güçlü *include* yönergesiyle derleme sırasında eklenir .

İşte her bağımsız program dosyasının, belirgin özelliklerinin açıklamasıyla birlikte bir tartışması. Dosyaların kendileri sonraki sayfalarda listelenmiştir. Devpac Amiga tarafından derlenmeye hazırdırlar, ancak sözdizimi kurallarına uyması için kılavuzda belirtilen değişiklikler yapıldığı takdirde herhangi bir 68000 derleyicisi kullanılabilir.

Bu, Amiga'nın tüm donanım ayrıntılarının bir ders kitabı olmadığından, çoğu zaman kendilerine ait tam bir bölümü hak eden birçok özelliğe yalnızca kısa değinilmesi kaçınılmazdır. Bu üzücüdür ancak anlatının akışını korumak ve genel hedefi gözden kaçırmamak için gereklidir. İlgilenen okuyucu, Amiga Donanım Referans Kılavuzu'nda tam bir donanım açıklaması bulacaktır.

3.4.1. *Polydraw'lar*

Bu ana kontrol programıdır. Başlangıçta *dahil edilen* diğer dosyalarda bulunan birçok alt rutini çağırır . Bu yönerge, montajcının montaj tamamlandığında programın bu noktasında başvurulan kaynak dosyasının tamamını eklemesini sağlar. Bu dosyada çağrılan tüm alt rutinler *core\_00.s* ve *systm\_00.s* bölümlerinde ayrıntılı olarak açıklanmıştır.

İlk bellek ekranlar ve bakır listeleri için tahsis edilir. Daha sonra Bakır listeleri oluşturulur, ardından blitter tahsisi ve renk tablosu yazılır. Ayrıca maske düzlemine erişimi hızlandırmak için yazılmış bir arama tablosu da vardır. Daha sonra program, yalnızca makineyi sıfırlayarak kesebileceğiniz sonsuz bir döngüye girer. Önce ekran 1'de, ilk mantıksal ekranda bir üçgen çizer ve başlangıçta boş olan ekran 2'yi görüntüler. Daha sonra ekran 1'de (şimdi fiziksel ekran) üçgeni görüntüler ve artık bir sonraki mantıksal ekran haline gelen ekran 2'de ters bir üçgen çizer. Artık döngünün sonuna ulaşılmıştır ve

\_sütyen blit\_döngüsü\_

döngü tekrarlanır. Tüm önemli işler, önerdiği şeyi yapan uzun alt rutin *poly\_fill'de yapılır — çokgenleri doldurur.*

Bu ana programdır. Çizilen şekilleri değiştirmeyi deneyebilirsiniz. Bunlar dolu çokgenlerdir ve köşelerinin koordinatları dosyanın sonunda verilmiştir. Bunlar hakkında daha sonra daha fazla bilgi vereceğiz.

Program basit ama bahsettiğimiz konuları anlatıyor ve ileride kullanacağımız tüm rutinleri içeriyor.

3.4.2.sistem\_00.s*​*

Bu bir "ev işleri" dosyasıdır; sıklıkla kullanılan yardımcı rutinleri içerir. Başlangıçta ekranlar ve Copper listeleri için bellek alanı ayırma rutini vardır; bu, Amiga İşletim Sisteminde özel Kütüphanelerde bulunan rutinlere yapılan çağrılarla yapılmalıdır. Amiga, ihtiyaç duyulduğunda boş alanın ayrıldığı dinamik bellek yönetimini kullanır. İlk 8 bitlik mikroların aksine, belirli bir bellek adresi aralığının programlar için ve başka bir aralığın da ekran için ayrıldığını varsayamazsınız. İşletim Sisteminden alan istemeniz ve ne elde edeceğinizi görmek için beklemeniz gerekir. Sistem ayırmak için bellek bulamazsa, *allocmem* çağrısından sonra d0 kaydında 0 döndürür . Program bu koşulu kontrol etmez ancak büyük programlarda muhtemelen olur ve dikkat edilmesi gereken bir şeydir.

Bizim amaçlarımız için ekranlar çip belleğinde olmalı ki özel donanım tarafından erişilebilsin. Bunun alt 512k aralığı olduğunu unutmayın (eğer bilgisayarınızda yalnızca bu kadar varsa o zaman hepsi çip belleğidir) ve istediğimiz alanın bu bellekte olduğunu belirtmek mümkündür. Yapabileceğimiz başka bir istek daha var; tahsis edildiğinde belleğin temizlenmesini isteyebiliriz — temiz bir sayfa ile başlamak faydalıdır. Bu isteklerin her ikisi de *allocmem* rutinini çağırmadan önce d1'e uzun kelime #$10002 yerleştirilerek yapılır. Buna daha detaylı bakmadan önce, kütüphanelerin nasıl kullanıldığına kısaca bakalım.

Kütüphaneler

Amiga'da birkaç kütüphane vardır. Bunlar, İşletim Sistemi tarafından "ücretsiz" olarak sağlanan benzer türden rutin kümeleridir. Düzenlenme biçimleri ve kütüphanelerde toplanmaları, sistemin C dil yönelimini yansıtır. Elbette bunları kendi rutinlerinizle değiştirebilirsiniz, ancak neden uğraşasınız ki (şüphesiz Amiga programcıları kütüphanelere bir dizi vektör grafik rutini eklemiş olsaydı bu kitap asla yazılmazdı).

Belirli bir kütüphane fonksiyonuna derleyicide erişim yolu oldukça basittir çünkü her alt rutin, kütüphanenin başlangıç ​​veya taban adresinden bellekte sabit bir mesafede (ofset) başlar (her kütüphanenin farklı bir taban adresi olacaktır). Bu nedenle bir kütüphane rutinini çalıştırmak için önce taban adresini alırsınız, sonra gerekli olan tüm giriş parametrelerini ayarlarsınız ve son olarak taban adresinden ofsete atlarsınız. Fonksiyon işini bitirdiğinde ana programa geri dönersiniz.

allocmem'de ekranlar ve Copper listeleri için bellek ayırmak amacıyla *Exec* adlı özellikle önemli bir kütüphane *kullanılır . Exec* , sıfırlamadan hemen sonra kullanılabilir olan işlevlere sahip bir tür Ana Kütüphanedir. Taban adresi her zaman aynıdır ancak her durumda 4 numaralı bellek adresinde uzun bir sözcük olarak saklanır ve a6 gibi bir kayıt defterine şu talimatla yerleştirilebilir:

hareket et.l 4,a6

*allocmem'de* olan da budur, ancak *equates.s'de* 4'e eşit olarak tanımlanan *execbase* sabiti daha okunabilir olduğu için kullanılır.

*Allocmem'e* baktığımızda, ilk önce Exec kütüphanesinin temel adresinin a6 kaydına yerleştirildiğini, ardından iki ekran düzlemini, maske düzlemini ve depolama düzlemini (arka planın çerez kesiminden önce depolandığı yer) oluşturmak için 12 bitlik düzlemlere eşit bir alanın d0 kaydına yazıldığını görebiliriz. Ardından, çip RAM'ini belirtmek için $2 ve tahsis sırasında belleği temizlemek için $10000'in birleşimi olan uzun $10002 sözcüğü d1'e yazılır. Son olarak, *allocmem* işlevine yapılan bir çağrı (temel adresten -198 bayt uzaklıkta) belleği tahsis eder ve ardından bu çeşitli parçalara bölünür. Son olarak, *allocmem işlevine yapılan daha fazla çağrı kullanılarak, her ekran için bir tane olmak üzere iki Copper listesi için alan tahsis edilir. Programı okunabilir kılmak için, ofsetlere equates.s* dosyasında tanımlandığı gibi çağrılacak işlevin adlarının verildiğine dikkat edin . Bir diğer ilginç özellik ise, işlev ofsetlerinin hepsinin negatif olmasıdır. Pozitif ofsetler, birçok önemli İşletim Sistemi değişkeninin saklandığı ExecBase yapısına yol açar.

Bunun ardından iki Copper listesi kurulur. Copper birkaç yararlı işlevi yerine getirebilir, ancak onu yalnızca dikey boşluk sırasında ekranları değiştirmek için kullanmayı planlıyoruz. İki Copper listesi, sistem düzlemi işaretçilerinin adreslerini, *bpl1pt* , *bpl2pt* ,.. *vb.* , gerçek ekran düzlemi adresleriyle yan yana, sırayla içerir. Copper listelerini oluşturmak için gereken tek şey budur. Donanım, etkin Copper listesi tarafından işaret edilen ekranı görüntüleme işini yapacaktır. Listelerin sonunda küçük bir numara vardır. Copper kendi başına bir işlemcidir ve talimatları yürütmek ister. Listelerin her birindeki son talimat (uzun $fffffffe sözcüğü), Copper'a ekran raster'ı imkansız y = $ff ve x = $fe konumuna gelene kadar beklemesini söyler. Kaydedebileceği en büyük yatay konum $e2 olduğundan bu imkansızdır. Bu nedenle yaptığı şey, asla gerçekleşmeyecek bir olayı beklemektir. Dikey boş kesme gerçekleştiğinde bu sıkıntıdan kurtuluruz, bu noktada bakır listelerini çift tamponlamayı uygulayacak şekilde değiştiririz.

*Blit-alloc'ta* blitter yalnızca bizim kullanımımız için ayrılmıştır ve görev değiştirme kapalıdır. Burada neler olup bittiğini tartışalım.

Blitter birçok işlevi yerine getirebilir; hatta disk sürücüsünü okumakla bile ilgilenir. Yoğun bir şekilde kullanmayı planladığımız için diğer tüm uygulamaları devre dışı bırakmak mantıklıdır. Bunu yapmak için grafik kütüphanesindeki *OwnBlitter işlevini kullanmalıyız. Burada bir sorun var çünkü ExecBase'in* aksine , diğer kütüphanelerin temel adresi sistem tarafından bilinmiyor; bulunması gerekiyor. Ancak diğer kütüphane adreslerini bulan *OpenLibrary adlı bir Exec kütüphane işlevi var. OpenLibrary,* a1 kaydındaki kütüphane adına bir işaretçi (bir ASCII dizesi yeterli olacaktır) ve d0'daki kütüphane sürümünü gerektirir. d0'daki kütüphane adresini (veya başarısız olursa sıfırı) döndürür; bu daha sonra kendi işlevleri için ofsetler için bir temel adres olarak kullanılabilir. Programda temel adres a6'ya yerleştirilir. Grafik kütüphanesinin temel adresini a6'da aldıktan sonra, *equates.s* dosyasında tanımlanan bir ofset olarak *OwnBlitter* işlevine atlamak ve blitter'ı yalnızca programımız için ayırmak kolaydır.

Çoklu görev

Amiga çoklu görev yapabilen bir makinedir. Tek bir CPU'su olmasına rağmen aynı anda birkaç program çalıştırıyormuş gibi görünebilir. Bu, her görevin kısa bir süreliğine açılıp bir sonraki sefere kadar tekrar kapatıldığı anlamına gelir. Her görev yavaşlatılır ancak genel etkisi çoklu görevdir. Bizim açımızdan çoklu görev gerçekleşmemelidir; isteyeceğimiz son şey başka bir uygulamanın veri yapılarını değiştirmesidir. Çoklu görev, *Exec* kütüphanesi işlevi *Forbid* ile kapatılır .

DMA

Alt rutin *init\_scrns'de* bir sonraki adım , oyun alanı yapısının, bütünlük için ilk mantıksal/fiziksel ekran atamasıyla birlikte kurulmasıdır (aslında ekran atamasının burada yapılması gerekmez çünkü zaten ana program döngüsünde ileri geri değiştirilecektir). Bu önemli veri yapılarına karışmadan önce, dışarıdan müdahalenin son bir kaynağı ortadan kaldırılır: DMA.

DMA, doğrudan bellek erişimi anlamına gelir; bu, sistemin çeşitli bölümlerinin 68000 işlemciyi aşırı yüklemeden bağımsız olarak belleğe okuma ve yazma yapmasına izin vermenin bir yoludur. Elbette, her şeyin aynı anda belleğe erişmesine izin verilemez ve genel kontrol, *DMACON* adlı ana kaydı aracılığıyla işlenebilen DMA denetleyicisi adı verilen ayrı bir işlemci tarafından yönetilir . *DMACON'un* iki ayrı bölümü vardır: biri salt okunur, diğeri salt yazılır. *DMACON'a* talimat göndermek istiyoruz , bu nedenle bizi ilgilendiren salt yazılır bölümdür. *DMACON'un* en yüksek biti (15) , kayda bir kelime yazıldığında özel bir işleve sahiptir: 1 ise yazılan bitleri ayarlar, 0 ise yazılan bitleri temizler. Yani üzerine $8004 yazarsanız 3. bit ayarlanır, ancak $0004 yazarsanız temizlenir. Sadece 3. bit bundan etkilenecek, diğer bitler bulundukları durumdan etkilenmeden kalacak. Sadece yazılabilir *DMACON'da* bit atamaları şunlardır:

BİT ADI İŞLEVİ

15 SET/CLR set/bitleri temizle

14 - işlev yok

13 - işlev yok

12 ve 11 - atanmamış

10 BLTPRI 1 ise, blitter 68000'e göre önceliğe sahiptir

9 DMAEN ana cihazı tüm bitler 0 - 8 için etkinleştirilir

8 BPLEN bit düzlemi DMA'yı etkinleştirir

7 COPEN Bakır DMA'yı etkinleştirir

6 BLTEN DMA blitter'ı etkinleştirir

5 SPREN sprite DMA'yı etkinleştirir

4 DSKEN disk DMA'yı etkinleştirir

3 - 0 AUDxEN ses DMA için kanalları etkinleştirir

*DMACON'a* ulaşmak için çip kayıt yapısının temel adresi olan $dff000, a5'e koyulur ve kayıta *equates.s'de* tanımlanan *DMACON* ofseti aracılığıyla erişilir . Oyun alanı ve ekran kurulurken tüm DMA, *DMACON'a* $03ff kelimesi yazılarak kapatılır .

Oyun Alanı

Sonraki oyun alanı başlatması, oyun alanı donanımıyla ilişkili bir dizi çip kaydına yazar. Düşük çözünürlükte beş bit düzlemi ve iç içe olmayan modda standart 200 x 300 boyutlu bir pencereyle çalıştığımız için, bu kayıtları ayarlamak olabildiğince basittir, ancak genel olarak kurulum oldukça karmaşık bir işlemdir.

Öncelikle oyun alanının ne kadarının görüntüleneceğine karar vermeliyiz çünkü burada ekrandaki görüntüden daha büyük bir oyun alanı seçeneği var. Ekrandaki görüntüye pencere adı verilir. Böyle bir seçenek kayan manzaraya sahip sprite grafiklerinde kullanışlı olurdu. Sadece oyun alanının ve pencerenin aynı boyutta olmasını istiyoruz. Buna göre yazılması gereken *DIWSTRT* (görüntüleme penceresi başlangıcı) ve *DIWSTOP* (görüntüleme penceresi sonu) adlı iki kayıt vardır. Buradaki önemli nokta, elektron ışını tüm monitör ekranını taramasına rağmen hepsine çizim yapmaması ve sisteme ekranın görüntüleme kısmının nerede başladığını ve bittiğini söylememiz gerektiğidir. Bu, resmin bozulduğu kenarları önlemek ve ayrıca boşluklar için yer bırakmak içindir. Başka bir bildirimde bulunmadan, düşük çözünürlük için standart değerleri kullanacağız: *DIWSTRT* = $2c81, *DIWSTOP* = $f4c1.

Pencere konumuna ek olarak sisteme penceredeki bit düzlemlerinden veriyi nasıl alıp görüntüleyeceğini söylemek gerekir. Bu bilgi *DDFSTRT* (veri alma başlangıcı) ve *DDFSTOP* (veri alma durdurma) kayıtlarında bulunur. Tekrar, daha fazla uzatmadan, normal düşük çözünürlüklü değerleri kullanacağız: *DDFSTRT* = $0038, *DDFSTOP* = S00d0.

*İş henüz bitmedi. Şimdi bit düzlemi kontrol kayıtları BPLCON0* , *BPLCON1* ve *BPLCON2* ile modül kayıtları *BPL1MOD* ve *BPL2MOD'u* ayarlamalıyız . Bunlardan gerçekten sadece BPLCON0 önemlidir çünkü renkli bit düzlemlerinin sayısını belirler. %0101001000000000 değerine ayarlanmıştır, bu da düşük çözünürlük (bit 15 = 0), 5 bit düzlemi (bit 12 ila 14), tutma ve değiştirme yok (bit 11 = 0), çift oynatma alanı yok (bit 10 = 0), renkli (bit 9 = 1), genlock sesi yok (bit 8 = 0), ışık kalemi yok, harici senkronizasyonun iç içe geçme modu (bit 1, 2 ve 3) anlamına gelir; 0 ve 4 ila 7 arasındaki bitler kullanılmaz. *BPLCON1* bizim ilgimizi çekmeyen kaydırma işlemlerini ayarlıyor ve *BPLCON2* ise bir kez daha istemediğimiz sprite'larla ilgileniyor.

BPL1MOD ve *BPL2MOD kayıtları bir oyun alanının dikdörtgen bir kesrini görüntülemekle ilgilidir ve çift ve tek sayılı bit düzlemleri için Modulo değerleri olarak adlandırılan değerleri içerir. Değerler modülü, sistemin dikdörtgen parçanın tüm oyun alanına göre nerede olduğunu* bilmesini mümkün kılar. Blitter'a geldiğimizde bu fikirle tekrar karşılaşacağız. Şu anda tüm oyun alanını kullandığımız için endişelenecek bir modül değeri yok.

*Son olarak, Copper listesi 1. ekrana yönlendirildiğinde, Copper, COPJMP1* kayıt defterine yazılarak açılır ve böylece bit düzlemi, Copper ve blitter DMA açılır.

Renkler

*systm\_00.s'deki* bir sonraki rutin renk paletini ayarlar. Bunun renk tablosunda bulunan 32 renk listesi (olası 4096'dan 32'si) olduğunu unutmayın. *data\_00.s* dosyasındaki *col\_tble'daki renk listesinin içeriğini COLOR00'dan* başlayarak 32 çip renk kayıtlarına aktarır . Renkler spektrumu takip eden tam bir settir ancak ilk renk arka plan olduğundan siyahla başlar.

*systm\_00.s* dosyasında kalanlar çizimle ilgilidir ve ekranları ve diğer düzlemleri silmek için bir dizi rutindir, bunları bir anlığına bir kenara bırakacağız ve iki tamamlayıcı ekran arabelleğe alma rutini *drw1\_shw2* ve *drw2\_shw1 . Bunların yaptığı şey, sistem kaydı COP1LC'yi* görüntülenecek bakır listesine yönlendirmek ve çizilecek ekranın adresini blitter için hazır bir şekilde *log\_screen* işaretçisine kaydetmektir.

*v\_blnk'in* en sonunda, monitördeki elektron ışını ekranın en üstüne geri uçtuğunda dikey boşluğu beklemek için bir rutin vardır. Aslında, resmin bozulmasının olacağı ekranın en üstünde çizim yapmaktan kaçınmak için çizilmemiş bir dizi çizgi vardır. Elbette bu, resmin en üstünü kaybettiğimiz anlamına gelmez, sadece resmin monitör ekranının bir kısmına gelene kadar başlatılmadığı anlamına gelir. Özel bir çip kaydı *VPOSR,* raster taramasının dikey konumunun bir kaydını tutar ve okuyarak nerede olduğunu bulabiliriz. Tarama en üstteki "gizli" bantta olduğunda ekranları değiştirerek değişikliği gizleyebiliriz. Yeterince gizli bir satır $10 numarasıdır. *v\_blnk'in* yaptığı şey, bu konuma ulaşana kadar beklemektir. *VPOSR,* aslında dikey konumu 8 ila 16 bit aralığında olan uzun bir kelime kaydıdır, yani iki kelimeyi kapsar, bu nedenle bu bitler tek tek seçilmelidir.

3.4.3. *çekirdek\_00.s*

Tüm eylemin gerçekleştiği yer burasıdır. Bu dosya, çizimi yapan önemli alt rutin *poly\_fil'i içerir.*

*Bu büyük kod bölümünde blitter kayıtlarına yazmadan önce blt\_chk* rutininin nasıl kullanıldığına dikkat edin. Bunun nedeni, blitter'ın 68000 CPU'dan bağımsız çalışması ve bir sonraki ihtiyaç duyulduğunda önceki görevi tamamlama olasılığının yüksek olmasıdır. Sonuç olarak, bitmesini beklemeliyiz. Hala devam ederken kayıt içeriğini değiştirmeye yönelik herhangi bir girişim, garip etkilere yol açacaktır.

Burada neler olup bittiğini kısaca özetleyelim. Rutin, belirli bir renkle doldurulacak bir poligonun köşelerinin koordinatlarını geçirir. Bunu yapmak için önce doldurulmaya uygun bir biçimde bir taslak çizmeli ve sonra hız için blitter kullanarak doldurmalıdır. Ayrıca blitter, bir sonraki kareye hazır hale getirmek için ekranı silmeye dahil olacaktır. Bunun için rutinler *systm\_00.s dosyasında olsa da, bir poligonu dolduran rutinlere çok benzerler. Aslında silmede ekran basitçe hiçbir şeyle doldurulur! poly\_fil'in* ne içerdiğine ayrıntılı olarak bakalım .

Anahattı Çizmek

Herhangi bir şeye başlamadan önce maske düzlemi silinir.

*Blitter'ın dolduracağı ana hatları çizen değiştirilmiş Bresenham algoritmasıyla zaten tanıştık. Bu, polyfil'in* 1. Bölümünde yapılan şeydir . İlk olarak ana hatların koordinatları hesaplanır ve geçici olarak *xbuf'ta* başlayan 200 uzun kelimeden oluşan RAM'in ayrılmış bir bölümünde (x-tamponu) saklanır . Her uzun kelime, maske düzlemindeki belirli bir y pozisyonuna atıfta bulunur ve yüksek ve düşük kelimelerinde ana hatların başlangıç ​​ve bitiş x-koordinatlarını tutar. Bu, tarama (y) satırı başına ikiden fazla piksel ayarlamaktan kaçınmanın basit bir yoludur. X-tamponu yalnızca bir başlangıç ​​ve bir bitiş x-değeri tutabilir ve bu da bir sondur!

Ana hat x-tamponunda birleştirildikten sonra, Bölüm 2'de maske düzlemine çizilir.

Blitter ile Doldurma

*Polyfil'in* 3. Bölümünde, maske düzleminde çizilen ana hat doldurulur ve ardından mantıksal ekranı oluşturan beş bit düzleminin her birine kopyalanır. Her bit düzleminde görünen şey, renge ve hangi mantığın dahil edildiğine göre belirlenir.

Maske düzlemindeki taslağı doldururken, blitter'ın dikkatini poligonu içeren bir dikdörtgene sınırlayarak zamandan tasarruf edebiliriz. Bu, özellikle yalnızca küçük nesneler çizildiğinde önemlidir. Blitter'ın boş olan ekran kısımlarına bakarak zaman kaybetmesini istemeyiz. Poligonun köşeleri dikdörtgeni oluşturmak için kullanılabilir.

Blitter kayıtlarının ayarları artık temel blitting süreci, yani verileri buradan oraya kopyalama açısından yorumlandığında biraz anlam ifade etmeye başlıyor. Blitter'ı tüm ihtişamıyla görmek için maske düzleminin ekran bit düzlemlerine eşlenmesinin tartışılmasını beklememiz gerekecek (sonraki).

Blitter doldurmanın ilginç bir özelliği geriye doğru çalışmasıdır; doldurma ekran koordinatlarında sağdan sola doğru azalan adresler sırasına göre gerçekleşir ve bu nedenle anahat üzerindeki en yüksek adres olan "sağ alt" tepe noktasından başlar. Bu adres iki blit işaretçisi A ve D'ye verilmelidir. Normalde basit bir blok görüntü transferinde bu işaretçiler sırasıyla kaynak ve hedef adreslerine işaret eder. Bu durumda bunlar aynı şeydir.

BLITMOD kayıtları *düzlem* genişliği ile blit dikdörtgeni arasındaki farkla ayarlanır. Bunlar blitter'ın her bir sonraki satırda doğru yerden başlamasını sağlar.

*BLITCON0* ve *BLITCON1* kayıtlarının içerikleri artık belirli anlamlara sahiptir. *BLITCON0'da* en üst bitler 12-15, asla kullanmadığımız kaydırma ile ilgilidir, bu nedenle $0 olarak ayarlanır. Bitler 8-11, DMA kanalları A, B, C ve D'yi açar. Doldurmanın sınırlı uygulamasında yalnızca A ve D kullanılır, bu nedenle nibble $9'dur. Bitler 0-7, kanalları belirli bir şekilde birleştiren mantık işlevini ayarlar. Bu durumda A'yı D'ye kopyalamak için $F0 olarak ayarlanır (bu mantık işlevlerinin ayarı aşağıda daha ayrıntılı olarak ele alınmaktadır). *BLITCON1'de* en üst bitler 12-15 yine kaydırma ile ilgilidir ve kullanılmaz. Bit 3, sınır çizgilerini içeren kapsayıcı bir doldurma için ayarlanır (bit 4, dışlayıcı olarak ayarlanır). Bit 1, azalan mod için ayarlanmalıdır. Kopyalanan nesnenin yanında bulunan bir kelime içindeki istenmeyen bitleri filtrelemek için maskeler içeren *BLTAFWM* ve *BLTALWM* adında iki kayıt daha vardır . Doldurma sırasında hepsi 1 olarak ayarlanır.

Blitter ile Kopyalama

İşte blitter'ın öne çıktığı yer burası. Yapmak istediğimiz şey, doldurulmuş çokgen şeklini maskeleme düzleminden mantıksal ekranı oluşturan beş bit düzlemine kopyalamak. Bunu nasıl yapacağımız hem çokgenin rengine hem de ekrandaki bileşik resmin nasıl bir araya getirildiğine bağlıdır. 32 kullanılabilir rengin, renk tablosunun 32 renk kaydına girilmiş kodları olduğunu unutmayın. 5 bit düzlemleri buradan gelir, çünkü 2^5 = 32'dir. İkiliye dönüştürüldüğünde kullanılacak renk kaydının numarası, rengini kullanmak için hangi bit düzlemlerinin ayarlanması ve hangi bit düzlemlerinin temizlenmesi gerektiğini söyler. Yani 7 numaralı kayıttaki rengi kullanmak istiyorsak, birinci, ikinci ve üçüncü bit düzlemleri 1 olarak ayarlanmalı ve dördüncü ve beşinci bit düzlemleri, görüntünün maskeleme düzleminde ayarlandığı bölgeler üzerinden temizlenmelidir. Blitter söz konusu olduğunda, bu beş blok görüntü transferi anlamına gelir ve bunun için yapılmıştır.

Ek karmaşıklık, mantıksal ekrandaki bileşik resmin bir araya getirilme biçiminden kaynaklanır. Uzak nesnelerin yakın nesneler tarafından gizlendiği bir tür sahne olacağından, uzak nesneleri önce, yakın nesneleri de sona koymalıyız. Bu nedenle, bir görüntüyü maske düzleminden bit düzlemlerine kopyalarken, halihazırda orada olanı bozmamaya dikkat edilmelidir, özellikle de kopyalanan şey, çokgeni içeren dikdörtgenin içindeki her şey olduğundan, çokgenin etrafındaki alan dahil (tüm bunlar şekil 3.2, 3.3 ve 3.4'te gösterilmiştir). Tüm bunların anahtarı , maskenin ve ekran hedefinin nasıl etkileşime girdiğini kontrol eden mantıksal işlemleri tanımlamanın yolu olan *minterms'dir* . Yapmak istediğimiz şey "çerez kesme" işlevidir. İşte nasıl çalıştığı.

Öncelikle maske düzleminde dolu bir çokgenin çizildiğini varsayalım. Şimdi çokgeni çevreleyen dikdörtgeni mantıksal ekranın ilk bit düzlemine kopyalayacağız. İlk renk biti ayarlanmışsa, maskede 1 olarak ayarlanan şey ilk bit düzleminde 1 olarak ayarlanmalıdır. Ancak maskede 0 olarak ayarlanan şey (yani çokgenin etrafındaki boşluk ancak tüm blitlenmiş dikdörtgenin içinde) bit düzleminde zaten olanı geçersiz kılmamalıdır. Öte yandan, ilk renk biti ayarlanmamışsa, maskede 1 olarak ayarlanan şeyin bit düzleminde 0 olarak ayarlanmasını isteriz. Ancak çokgenin etrafındaki, blitlenmiş dikdörtgenin kenarına kadar olan sınır, bit düzleminde zaten olanı değiştirmemelidir.

Şimdi yaptığımız şeyin bit düzlemindeki görüntüyü bit-bilge mantıkla maske görüntüsüyle birleştirmek olduğu oldukça açıktır. İşlemi, bir hedef kanal, son bit düzlemi üretmek için iki kaynak veri kanalının, maske ve bit düzleminin mantıksal bir kombinasyonu olarak düşünebiliriz. Blitter verileri bu şekilde işler. Dört DMA kanalı vardır - A, B ve C olarak adlandırılan üç kaynak ve D olarak adlandırılan bir hedef kanal. Bizim amacımız için, maske düzlemi A kanalında, orijinal bit düzlemi verisi B kanalında ve son kombinasyon D kanalındadır. C kanalı kullanılmaz. Çakışmayı önlemek için, bit düzlemi hem kaynak hem de hedef olduğundan, içindeki etkin dikdörtgen önce *depolama düzlemi* olarak kaydedilir . Yani gerçek mantıksal kombinasyonda depolama düzlemi olan kanal B'dir.

A, B ve D düzlemlerini etiketledikten sonra mantıksal kombinasyonda ne olmasını istediğimizi belirtmek artık kolay:

Renk biti ayarlanmışsa, D'deki (son bit düzlemi) bir bit, A'da (maske) VEYA B'de (depolama) ayarlanmışsa ayarlanmalıdır; diğer herhangi bir kombinasyon onu temizler: D = A VEYA B

Renk biti ayarlanmamışsa, D'deki bir bit, A'da ayarlanmamışsa VE B'de ayarlanmışsa ayarlanmalıdır; diğer herhangi bir kombinasyon onu temizler: D = (A DEĞİL) VE B.

Aslında hepsi bu kadar. Buna "cookie-cut" fonksiyonu denmesinin sebebi, sanırım maskenin kesilip bit düzleminin üstüne yerleştirilmesi. Mantıksal talimat, *BLITCON0* kaydının *LF* bitlerini (0 - 7 sayıları) uygun şekilde ayarlayarak verilir . Bu, mantıksal ifadeyi A, B ve C'yi içeren ürünler açısından genişleterek yapılır; bunların her birine *minterm* denir . Her *minterm'in kendisine ayrılmış bir LF* biti vardır; bu nedenle bu minterm genişletmede görünürse, *LF biti ayarlanır. Ayrıntılar bizi ilgilendirmez. Yukarıdaki mantık ifadeleri için LF* baytına girilecek değerler şunlardır:

ifade LF baytları

A VEYA B $FC

(A DEĞİL) VE B $0C

*Şimdi core\_00.s'ye* geri baktığımızda bunun gerçekleştiğini görebiliriz. İlk olarak bit düzlemindeki etkin dikdörtgen (maskedeki poligonu içeren) depolama düzlemine kaydedilir. Bu, mintermlerin *LF* baytı $F0'ı verdiği düz bir kopya olarak yapılır. *BLITCON0'daki* 8-11 bitleri , dört DMA kanalından hangisinin kullanıldığını sırayla belirtir; bu düz kopyada yalnızca A ve D.

Diğer blitter kayıtlarına yapılan girişler biraz yorum hak ediyor. *BLITxMOD* (x her kanal anlamına gelir) bit düzlem genişliği (40 bayt) ile geçerli blit dikdörtgeninin bayt cinsinden genişliği arasındaki farkı gerektirir ki bu elbette programa göre değişecektir. *BLITSIZE* üst on bitinde satır sayısını ve alt altı bitinde kelimeler cinsinden dikdörtgen genişliğini ister.

Daha sonra maskenin ve kaynak bit düzleminin mantıksal birleşimi, geçerli renk bitinin değerine bağlı olarak gerçekleşir. Prosedür, beş renk düzleminin her biri için tekrarlanır.

Son olarak blitter aynı zamanda *systm\_00.s* içindeki ekranları temizlemek için de kullanılır .

3.4.4. *eşitler.s*

Bu, donanım kayıtları, kütüphane fonksiyonları ve diğer sabitler için ofsetleri içerir.

3.4.5. *bss\_00.s*

Program içerisinde hesaplanan değişkenleri içerir.

3.4.6. *data\_00.s*

Bu, esas olarak standart paletin renklerini içerir.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Polydraw.s \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* SECTION TEXT

\* assembler directive

opt d+ put in labeles for debugging

bra main dont' try to execute the includes

\* all these files are to be include here

include equates.s all the constants

include bss\_00.s variables locations

include data\_00.s mainly standar palette colours

include systm\_00.s a lot of housekeeping routins

include core\_00.s the meat

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* heres the main control program

main

bsr alloc\_mem allocate memory for screens etc

bsr copr\_list set up the copper lists

bsr blit\_alloc take over the blitter

bsr colr\_set set up the standard palette

bsr wrt\_phys\_tbl look-up table for fast screen access

\* The program cycles here; screen buffering is used

blit\_loop:

\* first draw a triangle

bsr drw1\_shw2 draw on screen 1, display screen 2

lea my\_coords,a0 the vertices defined in the triangle

move.l a0,coords\_lst here's where to find them

move.w #2,colour coloured red

move.w #3,no\_in 3 sides to a triangle

bsr poly\_fill draw the outline and fill it red

\* then an inverted triangle

bsr drw2\_shw1 draw on screen 2, display screen 1

lea my\_inv\_coords,a0 the vertices

move.l a0,coords\_list here they are

move.w #12,colour coloured green

bsr poly\_fill draw the outline and fill it green

bra blit\_loop repeat the cycle

\* The coordinates of the two triangles

my\_coords dc.w 100,160,150,70,190,140,100,160

my\_inv\_coords dc.w 100,80,160,160,170,90,100,80

END

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Core\_00.s \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* This fills a polygon.

\* It consists of 4 parts:

\* 1. the x coords of the boundary are stored in xbuf

\* 2. the outline is drawn in the mask plane

\* 3. the the outline is filled by the blitter

\* 4. the blitter copies the mask to the bitplanes

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Part 1. Fill the buffer with the outline.

\* a3 pointer to crds\_in coords list (x1,y1...xn,yn,x1,y1)

\* a2 pointer to xbuf

\* d0-x1: d1-y1: d2-x2: d3-y2: d4-vertex number/decision vertex

\* d5-lowest y: d6-highest y/the increment: d7-edge counter

\* Polygon vertices are ordered anticlockwise

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

poly\_fill

bsr blit\_mask clear mask plane

\*INITIALISE ALL VARIABLES

filxbuf

move.w no\_in,d7

beq fil\_end quit if no more edges

move.l coords\_lst,a3

subq.w #1,d7 counter of num edges

move.w #MINIMUM\_Y,d5

clr.w d6 maximum y to zero

filbuf1

lea xbuf,a2

addq.w #2,a2 point to ascending side

move.w (a3)+,d0 next x1

move.w (a3)+,d1 next y1

move.w (a3)+,d2 next x2

move.w (a3)+,d3 next y2

subq.w #4,a3 point back to x2

\*FIND THE HIGHEST AND LOWEST Y VALUES: THE FILLED RANGE OF XBUF

cmp.w d5,d1 test(y1-miny)

bge filbuf3 miny unchanged

move.w d1,d5 miny is y1

filbuf3

cmp.w d1,d6 test(maxy-y1)

bge filbuf5 unchanged

move.w d1,d6 maxy is y1

filbuf5

exg d5,a5 save miny

exg d6,a6 save maxy

clr.w d4 init. decision var

moveq #1,d6 init. increment

\* All lines fall into 2 categories: [slope<1], [slope>1].

\* The difference is whether x and y are increasing or decreasing.

\* See if line is ascending [slope>0] or descending [slope<0].

cmp.w d1,d3 (y2-y1)=dy

beq y\_limits ignore horizontal altogether

bgt ascend slope > 0

\* It must be descending. Direct output to LHS of buffer. a2 must

\* be reduced and we have to reverse the order of the vertices.

exg d0,d2 x1 and x2

exg d1,d3 y1 and y2

subq.w #2,a2 point to left hand buffer

ascend

sub.w d1,d3 now dy is positive

\* Set up y1 as index to buffer

lsl.w #2,d1

add.w d1,a2

\* Check the sign of the slope

sub.w d0,d2 (x2-x1)=dx

beq vertical special case to deal with

bgt pos\_slope

\* It must have a negative slope but we deal with this by making the

\* increment negative.

neg.w d6 increment is negative

neg.w d2 dx is positive

\* Now decide if the slope is High (>1) or Low (<1).

pos\_slope

cmp.w d2,d3 test (dy-dx)

bgt hislope slope is > 1

\* Slope is < 1 so we want to increment x every time and then

\* check whether to increment y. If so this value of x must be saved

\* dx is the counter. Initial error D1=2dy-dx.

\* If last D -ve, then x=x=inc, dont record x, D=D+err1

\* If last D +ve, then x=x+inc, y=y+inc, record this x, D=D=err2

\* err1=2dy; err2=2dy-2dx

\* d0=x: d2=dx: d3=dy: d6=incx.

move.w d2,d5

subq.w #1,d5 dx-1 is the counter

add.w d3,d3 2dy=err1

move.w d3,d4 2dy

neg.w d2 -dx

add.w d2,d4 2dy-dx= D1

add.w d4,d2 2dy-2dx=err2

move.w d0,(a2) save first x

inc\_x

add.w d6,d0 x=x+incx

tst.w d4 what is the decision?

bmi no\_stk dont inc y dont record x

add.w #4,a2 inc y, record x. next buffer place

move.w d0,(a2) save this x

add.w d2,d4 update decision D=D=err2

bra.s next\_x

no\_stk

add.w d3,d4 D=D+err1

next\_x

dbra d5,inc\_x increment x again

bra y\_limits

\* The slope is > 1 so change the roles of dx and dy.

\* This time increment y each time and record the value of x after having done so.

\* Init error D1 = 2dx-dy

\* If last D -ve, then y=y+inc, D=D+err1, record x

\* If last D +ve, then x=x+inc, y=y+inc, D=D+err2, record x

\* err1=2dx, err2=2(dx-dy)

\* d2=dx: d3=dy: d6=inc: d0=x

hislope

move.w d3,d5

subq.w #1,d5 dy-1 is counter

add.w d2,d2 2dx=err1

move.w d2,d4 2dx

neg.w d3 -dy

add.w d3,d4 D1=2dx-dy

add.w d4,d3 2dx-2dy=err2

move.w d0,(a2) save 1st x

inc\_y

addq.w #4,a2 next place in buffer

tst.w d4 what is the decision

bmi same\_x dont inc x

add.w d6,d0 inc x

add.w d3,d4 D=D+err2

bra.s next\_y

same\_x

add.w d2,d4 D=D+err1

next\_y

move.w d0,(a2) save x value

dbra d5,inc\_y

bra y\_limits

\* The vertical line x is constant. dy is the counter

vertical

move.w d0,(a2) save next x

addq.w #4,a2 next place in buffer

dbra d3,vertical for all y

\* Restore the y limits

y\_limits

exg d5,a5

exg d6,a6

next\_line

dbra d7,filbuf1 do rest of lines (if any left)

\* This part ends with min y in d5 and max y d6

move.w d6,ymax

move.w d5,ymin

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* PART 2. Copy the xbuf to the mask plane.

\* Set up the pointer

lea xbuf,a0 base address of buffer

move.l maskplane,a1 base address of maak plane

lea msk\_y\_tbl,a2 mask plane y look up table

sub.w d5,d6 num pairs to set -1

move.w d6,d7 is the counter

beq fil\_end quit if all sides horizontal

move.w d5,d2 miny is the start

lsl.w #2,d5 4\*min y = offset into xbuf

add.w d5,a0 for the address to start

subq.w #1,d2 reduce initial y

poly2

addq #1,d2 next y

move.w (a0)+,d0 next x1

move.w (a0)+,d1 next x2

cmp.w d0,d1 test(x1-x2)

beq poly4 cant draw a line with one point

move.w d2,d5 pass y

bsr set\_pix set the 2 pixels

poly4

dbra d7,poly2 repeat for all y values

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* PART 3. Fill in the outline

\* Confine the blit to the rectangle (xmax-xmin)\*(ymax-ymin).

\* First xmax and xmin are recorded to define the rectangle.

bsr blt\_chk

frme

move.w no\_in,d7

subq.w #1,d7

movea.l coords\_lst,a3 here they are

move.w #MINIMUM\_X,xmin initialise xmin

clr.w xmax and xmax

x\_test

move.w (a3),d0 next x

cmp.w xmin,d0 test(x1-xmin)

bgt lnblit4 xmin unchanged

move.w d0,xmin this is x min

lnblit4

cmp.w xmax,d0 test(x1-xmax)

blt lnblit5 xmax unchanged

move.w d0,xmax this x is xmax

lnblit5

addq.l #4,a3 increment x pointer

dbra d7,x\_test for all x

\* Here's the fill blit. Several things must be found.

\* Calculate the address of the bottom rh corner of the rectangle

\* bltstrt contains its offset in the plane

move.w xmax,d0

lsr.w #SIXTEEN,d0 xmax/16

move.w d0,d2 save it

add.w d2,d2 \*2 = byte position in row

move.w ymax,d1

mulu #WIDTH,d1 row address

add.w d2,d1

ext.l d1

move.l d1,bltstrt save offset in the plane

\* address to start blit

movea.l maskplane,a0 plane base address

add.l d1,a0 plus offset is where blit starts

move.l #$dff000,a5

move.l a0,bltapt(a5) SOURCE

move.l a0,bltdpt(a5) DESTINATION

\* bltmod says how much of plane to blit

move.w xmin,d1

lsr.w #SIXTEEN,d1 xmin/16

sub.w d1,d0 xmax/16 - xmin/16

addq.w #1,d0 word width of window

move.w d0,bltwidth save it

move.w #WIDTH,d2

add.w d0,d0 width in bytes

sub.w d0,d2 blitmod

move.w d2,blitmod

move.w d2,bltamod(a5) SOURCE MODULO

move.w d2,bltdmod(a5) DESTINATON MODULO

\* set the control registers for a simple descending fill.

move.w #$09f0,bltcon0(a5) USE A&D D=A (no shift)

move.w #$000a,bltcon1(a5) INCLUSIVE FILL, DESCENDING

move.w #$ffff,bltafwm(a5)

move.w #$ffff,bltalwm(a5)

\* set the size and do the blit

move.w ymax,d0

sub.w ymin,d0

addq.w #1,d0

lsl.w #6,d0 set height

add.w bltwidth,d0 and width

move.w d0,blitsize sizeof blit

move.w d0,bltsize(a5) do the fill

\* PART 4.

\* Copy the mask to the screen bitplanes which must be set or cleared

\* depending on it's colour bit.

\* Only the smallest rectangle is blitted.

\* The mask is used in the cookie cut function:

\* If the colour bit is set, the masked region is set

\* If the colour bit is clear, the masked region is cleared.

pln\_cpy

bsr blt\_chk

move.w #DEPTH-1,d7 number of planes to blit

move.w colour,d6

move.w #0002,bltcon1(a5) COPY DESCENDING

move.w blitmod,d0

move.w d0,bltamod(a5)

move.w d0,bltdmod(a5)

move.w d0,bltbmod(a5)

IFD DOUBLE\_BUFFERING

move.l workplanes,a2 get address of planepointers list

ELSEIF

move.l showplanes,a2 get address of planepointers list

ENDC

; sub.l #WIDTH\*HEIGHT,a0 (ready to increment in next part)

; add.l bltstrt,a0 offset to draw at

nxtplane ;LOOP POINT

bsr blt\_chk

; add.l #WIDTH\*HEIGHT,a0 get next bitplane base address

move.l (a2)+,a0 get next address into a0

add.l bltstrt,a0 and add offset to start drawing at...

\* store the destination plane first, (copy to storeplane)

move.l a0,bltapt(a5) SOURCE

move.l storeplane,a1 destination

add.l bltstrt,a1 start position of rectangle

move.l a1,bltdpt(a5) in plane 6

move.w #$09f0,bltcon0(a5) straight copy

move.w blitsize,bltsize(a5) store destination plane

bsr blt\_chk

\* now mask region and set/clear as colour bit dictates

movea.l maskplane,a1 the mask

add.l bltstrt,a1 start here

move.l a1,bltapt(a5) A IS MASK

move.l storeplane,a1

add.l bltstrt,a1 offset

move.l a1,bltbpt(a5) B IS STOREPLANE

move.l a0,bltdpt(a5) DESTINATION

\* do we set or clear the masked region?

lsr.w #1,d6 get colour bit into carry flag

bcc bltclr bit is zero so clear masked region

\* we have to set the masked region

move.w #$0dfc,bltcon0(a5) NO SHIFT: USE A,B,D: D=A OR B

bra bltcopy

bltclr

\* clear region

move.w #$0d0c,bltcon0(a5) NO SHIFT: USE A,B,D: D=NOT A AND B

bltcopy

move.w blitsize,bltsize(a5) perform the required blit function

dbf d7,nxtplane do all the planes

\* done

fil\_end

rts

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Get pixel address and mask to set pixels in the mask plane which

\* mark start and end of a scan line .

\* d0=x1: d1=x2: d2=y1: a0=xbuf: a1=maskplane base: a2=msk y line tbl

set\_pix

lsl.w #2,d5 4\*y is offset in table

movea.l 0(a2,d5.w),a3 row address in mask plane

move.l a3,a4 save it

\* set pixel x1

move.w d0,d3 save x1

lsr.w #EIGHT,d0 byte num in row (/8)

adda.w d0,a3 the byte containing the pixel

andi.w #$0007,d3 pixel num in word

subi #7,d3

neg.w d3 bit to set

clr.w d0

bset d3,d0 this is a mask

or.b d0,(a3) set the pixel

\* set pixel x2

move.l a4,a3 restore row address

move.w d1,d3 save x2

lsr.w #EIGHT,d1 byte num in row (/8)

adda.w d1,a3 the byte containing the pixel

andi.w #$0007,d3 pixel num in word

subi #7,d3

neg.w d3 bit to set

clr.w d0

bset d3,d0 this is a mask

or.b d0,(a3) set the pixel

rts

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Get the screen address of a word

\* at a0=base: d0=x: d1=y:

scrn\_wrd

move.w #WIDTH,d2 plane width

mulu d1,d2 y\*width

add.l a0,d2 + base

lsr.w #SIXTEEN,d0 x/16

add.w d0,d0 word pos in row

ext.l d0

add.l d0,d2 address

rts

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* See if last blit is finished

blt\_chk

move.l #$dff000,a5

move.l d7,-(sp)

blt\_chk1

move.w dmaconr(a5),d7

btst.l #14,d7

btst.l #14,d7

bne blt\_chk1

move.l (sp)+,d7

rts

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* BSS\_00.s \*

\* Put all the variables you want to use in here \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* POLYGON VARIABLES

colour ds.w 1 current colour

no\_in ds.w 1 number of polygon vertices

xmin ds.w 1 limits

xmax ds.w 1 for

ymin ds.w 1 xbuf

ymax ds.w 1

coords\_lst ds.l 1

\* BITPLANE VARIABLES

maskplane ds.l 1 base addresses

storeplane ds.l 1

log\_screen ds.l 1

msk\_y\_tbl ds.l 200 scan line addresses

xbuf ds.l 200 x buffer

bltstrt ds.l 1

bltwidth ds.w 1

blitsize ds.w 1

blitmod ds.w 1

showplanes\_list ds.l DEPTH a list of pointers to each of the bitplanes per playfield

workplanes\_list ds.l DEPTH

IFD A500

scrn1\_base ds.l 1

scrn2\_base ds.l 1

cl1adr ds.l 1

cl2adr ds.l 1

cladr ds.l 1

oldcop ds.l 1

ENDC

gfxversion ds.w 1 lib version

vbi\_flag ds.w 1

show\_bitmap ds.l 1 BitMap structure pointers

work\_bitmap ds.l 1

showlist ds.l 1 Copperlist pointes

worklist ds.l 1

showplanes ds.l 1 VIDEO Ram pointers

workplanes ds.l 1

draw\_buffer ds.l 1 Bitplane sized memory to construct objects in

frame\_done ds.l 1 flag to indicate frame finished

File\_Handle ds.l 1

File\_Buffer ds.l 1

DosBase ds.l 1

GrafBase ds.l 1

IntuiBase ds.l 1

LowLevelBase ds.l 1

OldActiView ds.l 1

intHandle ds.l 1 V40 interrupt handle

colormap ds.l 1

ReturnMsg ds.l 1 For system to use when we quit

\* SYSTEM STRUCTURES I WANT TO USE.

EVEN

vblank ds.b IS\_SIZE ;store interrupt structure here.

EVEN

my\_view ds.b v\_SIZEOF

EVEN

my\_viewport ds.b vp\_SIZEOF

EVEN

my\_rasinfo ds.b ri\_SIZEOF

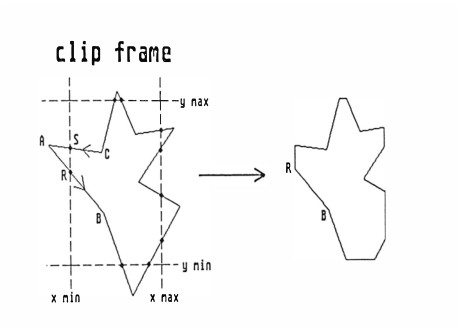
4. Pencereleme

Bir resim ekranın sınırlarından daha büyükse, fazlalığa ne olacağı konusunda bir sorun vardır. Bu olasılık için bir düzenleme yapılmadığı sürece, program ekran için ayrılmış RAM bölümünün dışındaki adreslere yazmaya çalışacaktır; bizim durumumuzda mantıksal ekran adı verilen beş bitlik düzlemlerdir. Her şeyin her zaman ekran boyutu içinde kalacağından emin olmadığımız sürece, resmin dışarıda kalan bölümlerini kesmek için bir düzenleme yapılmalıdır. Bir resmi bu şekilde sınırlamak, pencereden dışarı bakan birine benzetildiği için pencereleme olarak adlandırılır. Ekran, bilgisayarın iç dünyasına açılan bir penceredir. Bu pencere, belirli bir çözünürlükte izin verilen maksimum değer veya daha küçük bir şey olabilir (grafikleri hızlı hale getirmenin açık bir yolu, çok fazla çizilmesi gerekmeyecek şekilde resmi küçük tutmaktır). Görünür görüntünün boyutunu değiştirme özgürlüğü, özel efektlere bile yol açabilir; örneğin bir açılış açılışı. Resmin istenmeyen bölümlerinin 'kesilmesi' nedeniyle, bu pencerenin taslağına klip çerçevesi diyeceğiz.

İhtiyacımız olan algoritma, dolu poligonları işleyecek bir algoritmadır. Sadece kırpma çerçevesini aşan köşeleri kesmek yeterli değildir. Kesmenin bıraktığı çizgi poligonu kapatmak için ek bir kenar haline gelmelidir. Bu soruna zarif bir çözüm yine yıllar önce Sutherland ve Hodgman tarafından bulunmuştur.

4.1. Sutherland-Hodgman Kırpma Algoritması.

Sutherland-Hodgman algoritması aslında ihtiyaç duyduğumuzdan daha güçlüdür; her şekildeki çokgenleri işleyebilir. Bu kitapta, hız için yalnızca dışbükey (yuvarlak şekilli, tüm dış açılar sıfırdan büyük) çokgenler doldurulur. Dışbükey olma gereksinimi, daha sonraki bir kısıtlamanın sonucudur; gizli yüzey kaldırma algoritmasını basit tutma ihtiyacı. Bu, daha sonraki bir aşamada karşılaşacağımız bir şeydir.



*Şekil 4.1 Bir poligonun pencerelenmesi*

Kesin olarak konuşmak gerekirse, Sutherland-Hodgman çokgenlerin dışbükey olmasını veya kırpma çerçevesinin dikdörtgen olmasını gerektirmez. Ancak, basitlik adına, burada verilen versiyon monitör ekranıyla paralel dikdörtgen bir kırpma çerçevesi kullanır. Kırpma çerçevesinin sınırları xmin, xmax, ymin ve ymax ile tanımlanır ve Şekil 4.1'de genel bir çokgen için gösterilmiştir. Sutherland-Hodgman stratejisi, çokgenin tüm kenarlarının her sınırla kesişimlerini sırayla bulmaktır. Sınırımız dört kenarlı olduğundan, bu çokgenin dört döngüsünün yapılacağı anlamına gelir. Her döngüde orijinal kenarlardan bazıları kaybolabilir ve yenileri eklenebilir.

Her yeni tepe noktası incelendiğinde, onun ve önceki tepe noktasının konumuna bağlı olarak çeşitli eylemler gerçekleştirilir. Bu durumlar Şekil 4.1'de gösterilmiş ve aşağıda incelenmiştir:

1. Sonraki tepe noktası çerçevenin dışındaysa (A), önceki tepe noktasının konumunu kontrol edin, ©. Eğer o içerideyse, onları klip çerçevesiyle birleştiren kenarın kesişim noktasını (S) bulun ve kaydedin. Sonraki tepe noktasını (A) kaydetmeyin.
2. Sonraki tepe noktası çerçevenin içindeyse (B), önceki tepe noktasının (A) konumunu kontrol edin. Eğer dışarıdaysa, onları klip çerçevesiyle birleştiren kenarın kesişim noktasını bulun, ® ve kaydedin. Ayrıca sonraki tepe noktasını (B) kaydedin.

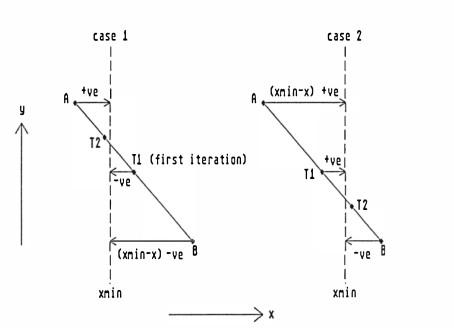
Bu, poligonun etrafındaki tüm köşelere uygulanan algoritmadır.

Bir kez daha, eğimli çizgilerin klips çerçevesiyle kesiştiği noktaları hesaplamanın bölme ve çarpma işlemlerini içeren çok sayıda matematiksel hesaplama gerektirdiği ortaya çıkabilir. Şaşırtıcı bir şekilde bu böyle değildir. Değişkenlerin soyut cebirsel semboller değil, bellek konumlarının veya kayıtlarının içerikleri olduğu montaj dili programlamada her zamanki gibi, yalnızca toplama ve çıkarma kullanarak ve ortaya çıktığı yerde, sağa ve sola kaydırmalarla hızlıca yapılabilen iki kuvvetlerine bölme ve çarpma kullanarak cevaplar bulmak mümkündür.

*Bunu açıklamak için, önceki noktanın xmin çerçeve sınırının dışında, ancak bir sonraki noktanın xmin* çerçeve sınırının içinde olduğu durumu ele alalım . Bu, Şekil 4.2'de daha ayrıntılı olarak gösterilmiştir; burada, limite en yakın olan noktaya bağlı olarak iki olası durum incelenir. B(x2,y2)'nin limitin içinde, A(x1,y1)'in ise limitin dışında olduğunu belirleme sürecinin bir parçası olarak, hem x1'i hem de x2'yi xmin ile karşılaştırmak gerekir. Ancak, yalnızca *COMPARE* komutunu kullanmak yerine, gerçek farklar (xmin-x1) ve (xmin-x2) hesaplanır ve sonucun işareti kararın temeli olarak kullanılır. (xmin-x1)'in pozitif, (xmin-x2)'nin negatif olduğunu unutmayın. Daha sonra belirlenecek ve kaydedilecek bir kesişim noktası olduğuna karar verildikten sonra, bu farklar kesişim noktasını aşağıdaki şekilde hesaplamak için başlangıç ​​noktası olarak kullanılır.

Kesişim noktasının koordinatlarından biri zaten biliniyor; xmin, yani sınırın kendisi; geriye kesişim noktasındaki y değerini bulmak kalıyor. Bu, aşağıdaki şekilde yinelemeli olarak yapılır. A ve B'nin ortalaması, koordinatlar toplanarak ve 2'ye bölünerek hesaplanır. Sonuç T1, A veya B'den kesişime daha yakındır ve (xmin-x1) ve (xmin-x2) ortalamasının işaretini takip ederek sınırın hangi tarafında olduğunu görebiliriz. Daha da önemlisi, (xmin-x1) ve (xmin-x2) ortalaması sıfırsa, y1 ve y2'nin ortalaması, kesişim değerinin kendisi olacaktır, çünkü bu olduğunda iki nokta sınırın her iki tarafında eşit aralıklarla yer alır veya onunla çakışır. Bu, örnek programda kullanılan yinelemeli algoritmanın temelidir.

İlk seferde olan şey, y1 ve y2'nin ortalamasının ve (xmin-x1) ve (xmin-x2)'nin ortalamasının bir toplama ve sağa kaydırma (ikiye hızlı bölme) yoluyla hesaplanmasıdır. Bu, T1 noktasının y koordinatlarını verir. (xmin-x1) ve (xmin-x2)'nin ortalaması sıfırsa, o zaman kesişim bulunmuştur. Eğer x ortalaması negatifse, 1. durumdaki T1 noktasında olduğu gibi, o zaman sınırın içinde yer alır ve bir sonraki ortalama (xmin-x1) ve (xmin-xT1) arasında alınmalıdır. Benzer şekilde, bir sonraki y ortalaması y1 ve yT1 arasında alınmalıdır. Öte yandan, (xmin-x1) ve (xmin-x2)'nin başlangıç ​​ortalaması pozitifse, durum 2'de olduğu gibi, bir sonraki ortalama (xmin-xT1) ve (xmin-x2) arasında ve bir sonraki y ortalaması yT1 ve y2 arasında alınmalıdır. Bu yinelemeli süreç, x ortalaması sıfır olana kadar devam eder, bu noktada geçerli y ortalaması, daha sonra kaydedilen kesişim noktasının y koordinatıdır.



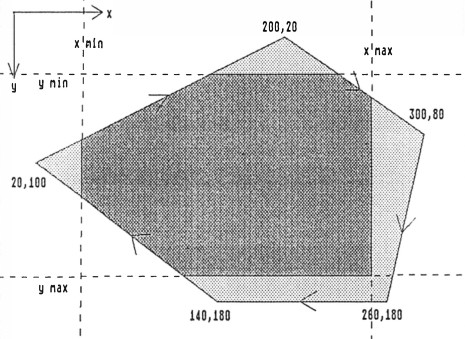
*Şekil 4.2 İterasyonla sınırın kesişimi*

4.2. Örnek Program

Örnek program, Sutherland-Hodgman algoritmasının bir versiyonunu kullanarak bir poligonu keser ve sonra onu doldurur. Poligon Şekil 4.3'te gösterilmiştir.

4.2.1.klip.çerçevesi.s

Bu, kontrol programı artı çokgen köşe noktaları için verilerdir. *my\_data'daki* koordinatlar , her zamanki gibi x0,y0,x1,y1…​…​x0,y0 sırasıyladır ve ilk koordinat en sonda tekrarlanır. Kırpma karesi sınırları da verilerde verilmiştir ve bunları kendinize göre değiştirebilirsiniz.



*Şekil 4.3 Pencereli çokgen*

4.2.2. çekirdek\_01.s

İşte gerçek kırpma rutini burada bulunur (şu ana kadar kullanılan diğer tüm rutinlerle birlikte, sonunda include *core\_00.s yönergesi aracılığıyla). İşin çoğu clip* alt rutini tarafından yapılır . Oldukça uzun görünüyor ancak bu onu daha okunabilir hale getirmeye çalışmaktır. Birçok parçası birbirine çok benzediği için, iç alt rutin çağrılarıyla daha kısa hale getirmek mümkün olabilir ancak o zaman takip edilmesi daha zor olur. Karmaşık bir rutindir ancak bu yukarıda açıklanan oldukça zor görevinin bir sonucudur.

Sınırlara göre kesildiği sırayla düzenlenmiştir: önce xmin, sonra diğerleri. Her dört verinin tam geçişi, her seferinde yeni köşeler eklenerek yapılır. Köşeler için veriler, *crds\_in'den* ilk geçişte girilir ve *crds\_out'a çıktı olarak verilir. Bir sonraki geçişte sıra tersine çevrilir. Dört geçiş olduğu için, veriler crds\_in'de* başladığı yere geri döner ve programın bir sonraki bölümüne hazır hale gelir, sonraki bölümlerde takip edilir.

4.2.3. bss\_01.s

Değişken sayısı arttıkça yeni bss dosyaları ortaya çıkar. Daha öncekilerin *dahil edilmesi* gerekir .

\* clipfrme.s

\*

\* Program for chapter 4

\* A program to clip and fill apolygon to a window (clip fram)

\* defined by the limits clp\_xmin, clp\_xmax, clp\_ymin, clp\_ymax

\*

\*SECTION TEXT

opt d+ incldue levels for debugging

bra main don't execute the includes

inlude equates. constants

inlude systm\_00.s housekeeping

include core\_01.s important subroutines

main bsr set\_up screens, copper, blitter, etc

blit\_loop:

bsr drw\_shw2

move.w #12-1,d7 six pari of coords for vertices

lea crds\_in,a0 destination

move.l a0,a3 ready for drawing

lea my\_data,a1 from here

clp\_loop

move.w (a1)+,(a0)+ transfer

dbf d7,clp\_loop them all

move.w #5,no\_in 5 sides to the polygon

move.w my\_colour,colour set the colour

move.w my\_xmin,clp\_xmin set the

move.w my\_max,clp\_xmax clip

move.w my\_ymin,clp\_ymin fram

move.w my\_ymax,clp\_ymax limits

bsr drw2\_shw1 draw on screen 2, display 1

bsr clip window it

bsr poly\_fill fill it

bsr drw1\_shw2 show the drawing

loop\_again:

bra loop\_again forever

\*SECTION DATA

\* A pentagon

my\_data dc.w 20,100,200,20,300,80,260,180,140,180,20,100

\* which is pink

my\_colour dc.w 24

\* The window limits

my\_xmin dc.w 50

my\_xmax dc.w 270

my\_ymin dc.w 50

my\_ymax dc.w 150

\*SECTION BSS

include bss\_01.s

\* SECTION DATA

include data\_00.s

END

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Core\_01.s \*

\* \*

\* A version of the Sutherland-Hodgman clipping algorithm.

\* It goes around the the polygon clipping it against one boundary at a time. It goes

\* around four times in all.

\* a0=crds\_in: a1=crds\_out: a2=no\_out: a3=saved(crds\_out):

\* d0=current limit: d1=x1: d2=y1: d3=x2: d4=y2: d5=(saved)x2: d6=(saved)y2:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

include core\_00.s

\* First clip against xmin.

clip

bsr clip\_ld1 set up pointers

tst.w d7 any sides to clip?

beq clip\_end not this time...

\* do first point as a special case.

move.w (a0)+,d5 1st x

move.w (a0)+,d6 1st y

move.w clp\_xmin,d0 limit

cmp.w d0,d5 test(x1-xmin)

bge xmin\_save inside limit

bra xmin\_update outside limit

\* do succesive vertices in turn

xmin\_next

move.w (a0)+,d3 x2

move.w (a0)+,d4 y2

move.w d3,d5 save x2

move.w d4,d6 save y2

\* now test for position

sub.w d0,d3 x2-xmin

bge xmin\_x2in x2 is in

\* x2 is inside, find x1

sub.w d0,d1 x1-xmin

blt xmin\_update both x2 and x1 are outside

\* x2 is out but x1 is in so find intersection, needs d1=dx1(+ve):d3=dx2(-ve)

\* d2=y1: d4=y2:

\* find the y intercept and save it.

bsr y\_intercept

\* but because it's out, don't save x2.

bra xmin\_update

xmin\_x2in

\* x2 is in but where is x1? GOD KNOWS!!

sub.w d0,d1 x1-xmin

bge xmin\_save both x1 and x2 are in

\* x2 is in but x1 is out so find intercept, but need -ve one in d3, so swap

exg d1,d3

exg d2,d4

bsr y\_intercept

xmin\_save

move.w d5,(a1)+ save x

move.w d6,(a1)+ save y

addq.w #1,(a2) inc count

xmin\_update

move.w d5,d1 x1=x2

move.w d6,d2 y1=y2

dbra d7,xmin\_next

\* The last point must be the same as the first

movea.l a3,a4 pointer to first x

subq #4,a1 point to last x

cmpm.l (a4)+,(a1)+ check first and last x and y

beq xmin\_dec already the same

move.l (a3),(a1) move first to last

bra clip\_xmax

xmin\_dec

tst.w (a2) if count

beq clip\_xmax is not already zero

subq.w #1,(a2) reduce it

\* Now clip against xmax. Essentially the same as above except that the order

\* of subtraction is reversed so that the same subroutine can be used to find

\* the intercept.

clip\_xmax

bsr clip\_ld2 set up pointers

tst.w d7 any to do?

beq clip\_ymin no...

\* do first point as a special case.

move.w (a0)+,d5 1st x

move.w (a0)+,d6 1st y

move.w clp\_xmax,d0

cmp.w d5,d0 test (xmax-x1)

bge xmax\_save inside limit

bra xmax\_update outside limit

\* do succesive vertices in turn

xmax\_next

move.w (a0)+,d3 x2

move.w (a0)+,d4 y2

move.l d3,d5 save x2

move d4,d6 save y2

\* now test for position

sub.w d0,d3

neg.w d3 xmax-x2

bge xmax\_x2in x2 is in

\* x2 is outside. where is x1?

sub.w d0,d1

neg.w d1 xmax-x1

blt xmax\_update both x2 and x1 are out

\* x2 is out but x1 is in so find intersection

\* needs dx1(+ve) in d1, and dx2(-ve) in d3, y1 in d2 and y2 in d4

\* find the intercept and save it.

bsr y\_intercept

\* but because its out dont save x2

bra xmax\_update

\* x2 is in but where is x1

xmax\_x2in

sub.w d0,d1

neg.w d1 xmax-x1

bge xmax\_save both x1 and x2 are in

\* x2 is in but x1 is out so find intercept

\* but must have the -ve one in d3,so switch

exg d1,d3

exg d2,d4

bsr y\_intercept

xmax\_save

move.w d5,(a1)+ save x

move.w d6,(a1)+ save y

addq.w #1,(a2) inc count

xmax\_update

move d5,d1 x1=x2

move d6,d2 y1=y2

dbra d7,xmax\_next

\* the last point must be the same as the first

movea.l a3,a4 pointer to first x

subq #4,a1 point to last x

cmpm.l (a4)+,(a1)+ check 1st and last x and y

beq xmax\_dec already the same

move.l (a3),(a1) move first to last

bra clip\_ymin

xmax\_dec

tst.w (a2) if count

beq clip\_ymin is not already zero

subq.w #1,(a2) reduce it

clip\_ymin

bsr clip\_ld1 set up pointers

tst.w d7 any to do?

beq clip\_ymax no...

\* do first point as a special case

move.w (a0)+,d5 ist x

move.w (a0)+,d6 1st y

move.w clp\_ymin,d0 this limit

cmp.w d0,d6 test (y1-ymin)

bge ymin\_save inside limit

bra ymin\_update outside limit

\* do successive vertices in turn

ymin\_next

move.w (a0)+,d3 x2

move.w (a0)+,d4 y2

move d3,d5 save x2

move d4,d6 save x1

\* now test for position

sub.w d0,d4 y2-xmin

bge ymin\_y2in y2 is in

\* y2 is outside where is y1?

sub.w d0,d2 y1-xmin

blt ymin\_update both y2 and y1 are out

\* y2 is out but y1 is in so find intersection

\* needs x1 in d1, x2 in d3, dy1 in d2 and dy2 in d4

\* find the intercept and save it

bsr x\_intercept

\* but because its out, dont save y2

bra ymin\_update

ymin\_y2in

\* y2 is in but where is y1

sub.w d0,d2 y1-ymin

bge ymin\_save both y1 and y2 are in

\* y2 is in but y1 is out so find intercept

\* but must have the -ve one in d4 so switch

exg d1,d3

exg d2,d4

bsr x\_intercept

ymin\_save

move.w d5,(a1)+ save x

move.w d6,(a1)+ save y

addq.w #1,(a2) increment no

ymin\_update

move d5,d1 x1=x2

move d6,d2 y1=y2

dbra d7,ymin\_next

\* the last point must be the same as the first

movea.l a3,a4 pointer to first x

subq.w #4,a1 point to last x

cmpm.l (a4)+,(a1)+ check first and last x and y

beq ymin\_dec already the same

move.l (a3),(a1) move first to last

bra clip\_ymax

ymin\_dec

tst.w (a2) if count

beq clip\_ymax is not already zero

subq.w #1,(a2) reduce it

clip\_ymax

bsr clip\_ld2

tst.w d7 any to do?

beq clip\_end no...

\* do first point as a special case

move.w (a0)+,d5 1st x

move.w (a0)+,d6 1st y

move.w clp\_ymax,d0

cmp.w d6,d0 test(ymax-y1)

bge ymax\_save

bra ymax\_update

\* do vertices in turn

ymax\_next

move.w (a0)+,d3 x2

move.w (a0)+,d4 y2

move d3,d5 save x2

move d4,d6 save y2

\* test for position

sub.w d0,d4

neg.w d4 ymax-y2

bge ymax\_y2in

\* y2 is outside where is y1?

sub.w d0,d2

neg.w d2 ymax-y1

blt ymax\_update both x2 and x1 are out

\* y2 is out but y1 is in so find intersection

bsr x\_intercept

bra ymax\_update

ymax\_y2in

\*y2 is in but where is y1?

sub.w d0,d2

neg.w d2 ymax-y1

bge ymax\_save both y1 and y2 are in

\* y2 is in but y1 is out so find intercept

exg d1,d3

exg d2,d4

bsr x\_intercept

ymax\_save

move.w d5,(a1)+ save x

move.w d6,(a1)+ save y

addq.w #1,(a2) increment num

ymax\_update

move.w d5,d1 x1=x2

move.w d6,d2 y1=y2

dbra d7,ymax\_next

\* the last point must be the same as the first

movea.l a3,a4 pointer to first x

subq.w #4,a1 point to last x

cmpm.l (a4)+,(a1)+ check first and last x and y

beq ymax\_dec already the same

move.l (a3),(a1) move first to last

bra clip\_end

ymax\_dec

tst.w (a2) if count

beq clip\_end is not already zero

subq.w #1,(a2) reduce it

clip\_end

lea crds\_in,a0

move.l a0,coords\_lst

rts

clip\_ld1

lea crds\_in,a0 pointer to vertex coords before

lea crds\_out,a1 and after this clip

move.l a1,a3 saved

move.w no\_in,d7 this many sides before

lea no\_out,a2 where the number after is stored

clr.w no\_out

rts

clip\_ld2

lea crds\_out,a0 pointer to vertex coords before

lea crds\_in,a1 and after this clip

move.l a1,a3 saved

move.w no\_out,d7 this many sides before

lea no\_in,a2 where the number after is stored

clr.w no\_in

rts

y\_intercept

tst.w d1

beq yint\_out

tst.w d3

beq yint\_out

movem d5/d6,-(sp)

yint\_in

move.w d2,d6

add.w d4,d6

asr.w #1,d6

move.w d1,d5

add.w d3,d5

asr.w #1,d5

beq yint\_end

bgt yint\_loop

move d5,d3

move d6,d4

bra yint\_in

yint\_loop

move d5,d1

move d6,d2

bra yint\_in

yint\_end

move.w d0,(a1)+

move.w d6,(a1)+

addq.w #1,(a2)

movem (sp)+,d5/d6

yint\_out

rts

x\_intercept

tst.w d2

beq xint\_out

tst.w d4

beq xint\_out

movem d5/d6,-(sp)

xint\_in

move d1,d5 x1

add.w d3,d5 x1+x2

asr.w #1,d5 ()/2=,x> a possible intercept

move d2,d6 dy1

add.w d4,d6 dy1+dy2

asr.w #1,d6 (dy1+dy2)/2 =<dy>

beq xint\_end if <dy>=0. boundry reached

bgt xint\_loop if not loop again

move d6,d4 unless <dy> is -ve and becomes dy2

move d5,d3 and <x> becomes x2

bra xint\_in and try again

xint\_loop

move d5,d1 <x> is new dx1

move d6,d2 and <dy> is new dy1

bra xint\_in

xint\_end

move.w d5,(a1)+ store intercept <x>

move.w d0,(a1)+ and the y as new vertex coords

addq.w #1,(a2) and increment the vertex count

movem (sp)+,d5/d6

xint\_out

rts next vertex

\* Leaves with a list of vertex coords at coords\_in

\* the number of polygon sides at no\_in

set\_up:

\* set up memory, screens, blitter etc

bsr alloc\_mem

bsr copr\_lst

bsr blit\_alloc

bsr colr\_set

bsr wrt\_phys\_tbl

include core\_00.s add on the previous core

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* BSS\_01 \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

include bss\_00.s

\* Polygon attributes

crds\_in ds.w 100 input coords

crds\_out ds.w 100 output as above

no\_out ds.w 1 output number

colr\_lst ds.w 20 list of polygon colours

clp\_xmax ds.w 1 clip frame limits

clp\_xmin ds.w 1

clp\_ymin ds.w 1

clp\_ymax ds.w 1

5. Olayları Perspektif İçerisine Yerleştirmek

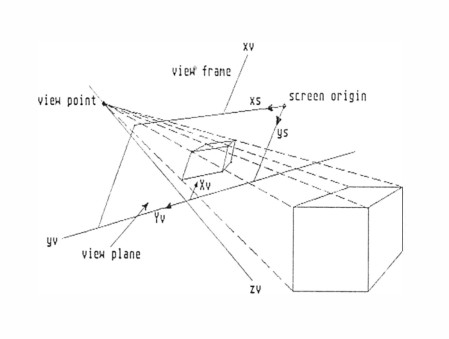
Uzaktaki nesnelerin yakındakilerden daha küçük görünmesi ilginç bir şeydir. Daha küçük değillerdir, ancak göze daha küçük bir açı verirler. Dolayısıyla herhangi bir sahnenin gerçek görünmesi için, ilkel nesnelerin boyutları uzaklaştıkça küçülmelidir. Tüm bunlar göz ve beyin tarafından yapılır. Perspektif dönüşümünün amacı, aynı etkiyi bilgisayar ekranında simüle etmektir.

Bu kitaptaki dönüşümleri kullanmak için aslında çok fazla matematik bilmenize gerek yok. Matematik ve dönüşümler hepsi çözüldü; sadece bunlara nasıl veri gireceğinizi anlamanız gerekiyor. Perspektif dönüşümü tam da böyle bir örnek. Ancak dönüşümleri tam olarak anlamak ve kullanmak için biraz matematik ve matris anlayışına sahip olmak gerekir. Bunları ihtiyaç duyuldukça tanıtacağız. Ekler ayrıca bu konularla ilgili bilgiler içerir

5.1. Perspektif Dönüşümü

Perspektif dönüşümü, bir nesnenin görüntüsünü dünya referans çerçevesinden ekrana yansıtan bir dizi matematiksel işlemdir. Bu, bir gölgenin oluşma şekline benzer, ancak bu durumda gölge nesnenin arkasına düşer ve daha büyük olur, perspektif projeksiyonunda ise bakış açısı ile ekran arasında ve daha küçüktür. Bu, Şekil 5.1'de gösterilmiştir.

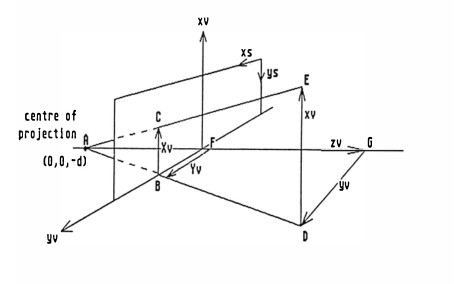
Dönüşümlerde ve matrislerde tekrar tekrar ortaya çıkan bir husus, homojen koordinatların kullanılmasıdır. Yine de bunları tamamen kullanmaktan kaçınmak mümkündür ve birçok durumda bunları hiç kullanmak sakıncalıdır. Bunlar ne anlama geliyor? Önemli mi? Bu bölümde homojen koordinatlar ve bunların yalnızca yöntemi göstermek için matris çarpımı kullanılarak yapılan perspektif dönüşümünde nasıl kullanılacağı hakkında bilgi ediniyoruz. Aynı zamanda, matris çarpımı kullanılmadan dönüşümün nasıl yapılacağı da açıklığa kavuşacaktır. Perspektif dönüşümünün bunu denemek için iyi bir fırsat olduğu ortaya çıkıyor.



*Şekil 5.1 Bir küpün perspektif izdüşümü*

Şekil 5.1, dünya çerçevesindeki bilgisayarın içinde tanımlanmış ve ekrana yansıtılmış bir nesneyi, bu durumda bir küpü göstermektedir. Ekran, görünüm çerçevesinin xv-yv düzleminde yer alır ve yansıtılan görüntü, görünüm noktasından gelen 'ışınların' (ayrıca zv ekseni boyunca -d'de yansıtma merkezi olarak da adlandırılır) görünüm düzlemini deldiği noktalarla tanımlanır. Pencere, ekranda görülebilen görünüm düzleminin alanıdır. Gerçekten hepsi bu kadar. Görünüm noktası bu şema içinde çok önemli bir rol oynar ve herhangi bir yere yerleştirilebilir. -z ekseni boyunca yerleştirilmesi cebiri basitleştirir ve izdüşümü görünüm çerçevesi kökenine göre ortalar. Bu çok basit bir izdüşüm türüdür; çizerler başka birçok tür kullanır. Ancak gayet iyi çalışır ve bununla ilişkili cebir minimumdur.

Hayatı kolaylaştırmak için, pencerenin monitör ekranını tamamen doldurduğu durumu ele alalım. O zaman ikisi arasındaki ayrım ortadan kalkar. Çok basit bir nesnenin ekrana nasıl yansıtıldığına bakalım. Bu, Şekil 5.2'de gösterilmiştir. Dönüşümün bir parçası olarak, orijinin sol üst köşede olduğu ekran koordinat sistemine de uyum sağlamak gerekir. Diyagramda gösterilen üç koordinat sistemi vardır: görünüm çerçevesi (xv,yv,zv), ekran çerçevesi (xs,ys) ve yansıtılmış koordinatlar (Xv,Yv). Bu yansıtılmış koordinat sistemi, kolaylık sağlamak için sunulan ve görünüm çerçevesi orijininde merkezlenen bir ara sistemdir.



*Şekil 5.2 Bir çizginin perspektif izdüşümü*

Benzer üçgenler ABC ve ADE ile benzer üçgenler ABF ve ADG'den şu sonuçları elde ederiz:

Xv/xv = d/(zv+d) ve Yv/yv = d/(zv+d)

veya

Xv = xv.d/(zv+d) ve Yv = yv.d/(zv+d).

Geriye sadece projeksiyonun görünür ekranda nereye merkezleneceğini seçmek kalır. Eğer ekranın alt kısmında yarı yolda merkezlenecekse, o zaman ekran koordinatlarında,

xs = Yv+Wx/2 ve ys = Wy-Xv

Burada Wx ve Wy, mevcut çözünürlükteki ekranın genişliği ve yüksekliğidir.

Düşük çözünürlükte Wx=320 ve Wy=200. Aşağıda sadece düşük çözünürlüğü ele alacağız, ancak bir çözünürlükten diğerine dönüşüm basittir.

Düşük çözünürlükte, ekran koordinatlarında görüntüleme için perspektif dönüşümü şu hale gelir:

xs = 160+yv.d/(zv+d) ys = 200-xv.d/(zv+d)

Bu dönüşümler basit cebir kullanılarak hesaplanabilir. Dikkat edilmesi gereken tek şey, paydanın asla sıfır olmamasıdır çünkü bu bir 'sıfıra bölme' istisnasına neden olur. Program buna dikkat edecek şekilde ayarlanabilir.

5.2. Homojen Koordinatlar

Yukarıdaki perspektif dönüşümü oldukça basittir ancak birkaç başka dönüşüm türüyle birleştirilecekse ciddi bir dezavantajı vardır. Unutmayın, matris dönüşümlerinin jargonunda birleştirme basitçe matrisleri birbiriyle çarpmak anlamına gelir. Dönüşümleri matris olarak yazmanın avantajı budur. Birkaç dönüşüm (döndürme vb.) ardışık olarak gerçekleştiğinde, genel dönüşüm tek tek dönüşümleri çarparak ve ardından tek seferde koordinatlara uygulayarak oluşturulabilir. Bu perspektif dönüşümündeki sorun, olduğu haliyle bir matris olarak yazılamamasıdır.

Temel olarak, bir matris doğrusal olan herhangi bir dönüşümü temsil edebilir, bu da başlangıç ​​ve dönüştürülmüş koordinatlar arasında orantılı bir ilişki olduğu anlamına gelir. Xv,Yv ve xv,yv,zv arasındaki dönüşümler için görmek istediğimiz şey şu denklemlerdir:

Xv = a.xv + b.yv + c.zv

Yv = d.xv + e.yv + f.zv

Burada a,b,c,d,e ve f katsayıları basit sayılardır.

Daha sonra bir matris ürünü olarak yazılabilir ( matrisler hakkında daha fazla bilgi için Ek 6'ya bakın)

XV ABC XV

= \* yd

Yv def zv

Ne yazık ki türettiğimiz perspektif dönüşümü bu forma sahip değil. Bunu ortaya koyan şey paydalardaki (zv+d)'dir; koordinatların kendileri paydalarda olmak zorundadır. Bu nedenle, şu anki haliyle dönüşümümüz 3x3 matris formuna konulamaz. Perspektif dönüşümü bu sorundan muzdarip olan tek dönüşüm değildir. Basit çeviriler de aynı şekilde muzdariptir. Sorundan kurtulmanın yolu homojen koordinatlara gitmektir.

Bizim açımızdan homojen koordinatların kullanımı bu sorunu aşmak için bir hiledir. Hile, daha fazla "alan" sağlamak için geçici olarak başka bir boyut eklemektir. Bu ek boyutun yaptığı tek şey budur çünkü bu ek boyutta tüm köşeler aynı değere, 1'e sahiptir. Homojen koordinatlarda (xv,yv,zv) noktası (xv,yv,zv,1) olur.

Bu nasıl yardımcı olur? Şimdi dönüşüm bir ürün olarak yazılabilir ancak ödenecek cezalar vardır: matris ürünü diğerlerine bölünmesi gereken ekstra bir terim üretecektir. Ayrıca tüm matrisler artık daha büyüktür (4x4). İşte nasıl çalıştığı.

Öncelikle ara sonucu elde etmek için perspektif dönüşümünü homojen koordinatlara yapalım:

d.xv d 0 0 0 xv

d.yv 0 d 0 0 yv

0 = 0 0 0 0 \* zv

zv+d 0 0 1 d 1

Daha sonra dördüncü elemana (zv+d) bölünerek elde edilir

Xv = xv.d/(zv+d)

Yv = yv.d/(zv+d).

Son olarak ekran merkezine çevirelim (bu çeviri homojen koordinatlarda matris çarpımı olarak da yapılabilir ama bu sadece iş yapmak olurdu):

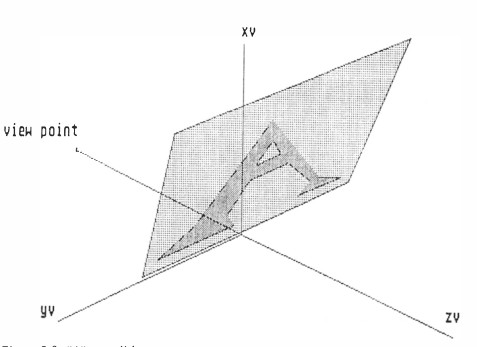
xs = 160 + yv.d/(zv+d)

ys = 200 - xv.d/(zv+d).

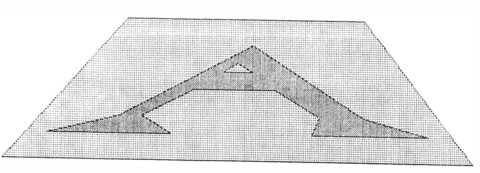
Perspektif matrisi, elemanlarının çoğu için sıfırlara sahiptir ve bu yüzden çarpmaların çoğu zaman kaybıdır. Dönüşümü gösteren bu bölümün sonundaki programda, homojen formu kullandık. Bu, montaj dilinde matris çarpımına faydalı bir giriş görevi görür ve bize birkaç az kullanılan montajcı talimatını deneme olanağı sağlar.

5.3. Örnek program

Örnek program, dünya çerçevesinde öne doğru eğimli "A" harfine (bir A monoliti) sahip bir uçağın görünümünü gösterir. Perspektif dönüşümü yapıldığında (pencereleme ve diğer her şeyle birlikte) ekranda bir filmdeki açılış logosu gibi görünür, kelimeler mesafeye doğru azalır. Şekil 5.3, uçağın görünüm çerçevesinde nasıl kurulduğunu gösterir. Şekil 5.4, ekranda nasıl göründüğünü gösterir.



*Şekil 5.3: "A" monolit*



*Şekil 5.4: Bir monolitin ekran görüntüsü*

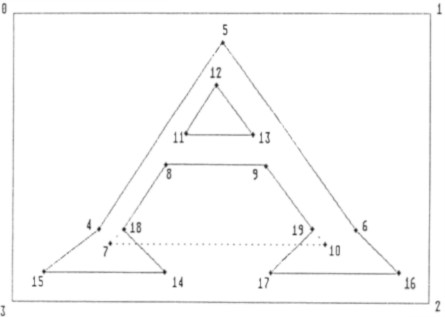
Veri dosyasındaki koordinatlara bakabilir ve farklı yönlerde nasıl göründüğünü görmek isterseniz değiştirebilirsiniz. İsterseniz, farklı bir şey çizmek için verileri tamamen değiştirebilirsiniz, ancak önce verilerin nasıl düzenlendiğini dikkatlice okuyun. Bu, veri dosyasında aşağıda daha ayrıntılı olarak açıklanmıştır. Karakterleri birleştirmeye ve köşeleri düzgün bir şekilde etiketlemeye dikkat edin.

5.3.1. perspektifler

Bu kontrol programıdır. İşlevi, verileri yüklemek, resmi çizmek ve bir tuşa basarak bitirmektir. Veriler aşağıda açıklanan *data\_01.s dosyasında saklanır.*

5.3.2. data\_01.s

Bu, veri listelerini içerdiği için daha sonra tartışılacaktır. Bunların nasıl kullanıldığını anlamak, programın nasıl çalıştığını anlamak için önemlidir. Görünüm çerçevesinde 3B olarak çizilmiş bir nesneyle başladığımızdan, köşelerinin her biri üç koordinatla (xv,yv,zv) sabitlenmelidir. Bunların listeleri *my\_datax* , *my\_datay* ve *my\_dataz* adreslerinde tutulur . Bu listelerdeki her köşeyi tanımlamak için bir şema vardır. Her köşenin, Şekil 5.5'te gösterildiği gibi bir numarası vardır. Koordinatlarını bulmak için, ilk koordinatı sıfır sayısı olarak sayarak baştan okuyun. Her poligondaki köşe sayısı vectors adresinde verilir.



*Şekil 5.5: Bir monolitin tepe noktası numaraları*

Resmi gerçekten çizmek için bundan daha fazla veri gerekir. Köşeler arasındaki bağlantılar *my\_edglst* içinde belirtilir . Her çokgen için bu tabloda bir bağlantı listesi vardır. Genel nesne, hepsi aynı düzlemde bulunan 6 çokgene bölünmüştür. Bunlar için köşe bağlantıları, saat yönünde giderek ve çokgeni kapatarak,

çokgen 0: 0,1,2,3,0

çokgen 1: 4,5,6,4

çokgen 2: 7,8,9,10,7

çokgen 3: 11,12,13,11

çokgen 4: 14,15,4,18,14

çokgen 5: 16,17,19,6,16

Bu şekilde düzenlendiğinde, nesneyi çizmek için gereken tüm bilgiler kolayca kullanılabilir. Çokgenleri renklendirmek için *my\_colour* adresinde ayrı ayrı renklerin bir listesi tutulur . Bu resimde, her bir parçayı ayrı ayrı çizmek yerine, bir anahat çizerek (çokgen 1) ve açık kısımları (çokgenler 2 ve 3) arka plan rengiyle maskeleyerek "A" nın oluşturulmasına karar verildiğine dikkat edin. Bu, arka planın altın ve harflerin macenta olduğu görülebilen gerçek renk listesi *my\_colour'dan* da açıkça anlaşılmaktadır . Bunu bu şekilde yapmak biraz zaman kazandırır ancak sınırlar tam olarak eşleşmediğinde sorunlara yol açabilir. Bu listeleri tamamlamak için toplam çokgen sayısı *my\_npoly* adresinde verilir . Bunlar, başlatma sırasında yüklenmesi gereken veri bloklarıdır. Diğer değişkenler, program ilerledikçe programın çeşitli bölümleri tarafından hesaplanır.

Dilediğinizi çizmek için bu listeleri değiştirebilirsiniz. Sadece bunun görünüm çerçevesindeki bir 3B nesne olduğunu ve koordinatların farklı eksenler boyunca görünümlerden belirlenmesinin en kolay olduğunu unutmayın. Ayrıca Şekil 5.3'te gösterildiği gibi görünüm düzleminin önüne yerleştirilmelidir.

5.3.3. data\_02.s

Perspektif dönüşümü için 4x4 matrisi burada, her seferinde bir satır, görünüm çerçevesi z ekseninde -100'de bir bakış açısıyla saklanır. Bu dosya yalnızca bir kez kullanılacağı için *data\_01.s'ye* dahil edilmez

Dönüşümde kullanılan matris çarpımını takip edemiyorsanız endişelenmeyin. Dönüşümü bir işlevi yerine getirmek için bir 'makine' parçası olarak düşünün. Görüş açısını değiştirmek istiyorsanız, 100 sayılarının her birini görüş noktasının yeni konumuna değiştirin. Burada 100'ün negatif zv ekseni boyunca görüş noktasının mesafesi olduğunu unutmayın.

5.3.4. bss\_02.s

Bu, programlar tarafından kullanılan değişkenlerin bir listesini içerir. Veriler, kontrol programı tarafından *data\_01.s* veri dosyasından değişken bloklarına yüklenir . Kontrol programından, neyin nereye gittiği açıktır. Görünüm çerçevesindeki köşelerin x, y ve z koordinatlarının listelerinden ve önceki bölümlerde açıklanan diğer özniteliklerden oluşur.

5.3.5. çekirdek\_02.s

Bunun iki kısmı var: Perspektif dönüşümü ve kırpma ve gerçek çizimle ilgilenen *polydraw .*

Perspektif dönüşümü homojen koordinatlarda matris çarpımı ile yapılır. Doğrudan cebir ile yapılabilir ancak homojen koordinatların ve matris çarpımının kullanımını çok kompakt bir şekilde göstermek için bu şekilde yapılır. Ayrıca yararlı ancak az kullanılan bir derleyici talimatı olan *LINK'i* kullanır . Çağrıldığında, işlemcinin yığında çerçeve adı verilen bir alan açmasına neden olur; burada veriler ana yığınla çakışmadan saklanabilir. Adres kayıtlarından biri olan çerçeveye işaretçi, gerekli alanla birlikte *LINK talimatında bildirilir. İşlemci, düzenli yığın işaretçisini çerçeveden uzakta ayarlamakla ilgilenir. Mevcut durumda, ara perspektif hesaplamalarının saklandığı yerdir. İş tamamlandığında, çerçeve UNLK* talimatı aracılığıyla kapatılır ve yığın işaretçisinin düzenlenmesi işlemci tarafından yapılır.

Perspektif dönüşümü, köşelerin görünüm düzlemindeki izdüşümlerini hesaplar ve bunları iki listede depolar: *scoordsx* ve *scoordsy* .

Polydraw son kısımdır. Çizimi tamamlamak için gerekli olan tüm önceki alt rutinleri içerir. Ayrıca başlangıçta her poligonun görünürlüğü için bir test içerir. Bu, gelecek şeylerin beklentisidir. Test, $1f'den büyük bir renk numarası aramaktır. Poligonun bakış noktasından uzağa baktığı bulunursa böyle bir değer daha önce ayarlanmış olurdu.

\*

\* perspect.s

\*

\*SECTION TEXT

opt d+ labels for debugging

bra main dont execute the includes

include core\_02.s core subroutines

include systm\_00.s

main bsr set\_up allocate memory etc

\* Transfer data from the data file to variables locations:

\* first the edge numbers and colours

move.w my\_npoly,d7 no of polygons?

beq main if none, quit

move.w d7,npoly or becomes

subq.w #1,d7 the counter

move.w d7,d0 save it

lea my\_nedges,a0 source

lea snedges,a1 destination

lea my\_colour,a2 source

lea col\_lst,a3 destination

loop0 move.w (a0)+,(a1)+ transfer edge nos

move.w (a2)+,(a3)0 transfer colours

dbra d0,loop0

\* second the edge list and coordinates

move.w d7,d0 restore count

lea my\_nedges,a6

clr d1

clr d2

loop1 add.w (a6),d1

add.w (a6)+,d2

addq #1,d2 last one repeated each time

dbra d0,loop1 = total no of vertices

subq #1,d2 the counter

lea my\_edglst,a0 source

lea sedglst,a1 destination

loop2 move.w (a0)+,(a1)+ pass it

dbra d2,loop2

move.w d1,vncoords

subq #1,d1

lea vcoordsx,a1

lea my\_datax,a0

lea vcoordsy,a3

lea my\_datay,a2

lea vcoordsz,a5

lea my\_dataz,a4

loop3 move.w (a0)+,(a1)+

move.w (a2)+,(a3)+

move.w (a4)+,(a5)+

dbra d1,loop3

\* the clip form boundaries

move.w my\_xmin,clp\_xmin ready

move.w my\_xmax,clp\_xmax for

move.w my\_ymin,clp\_ymin clipping

move.w my\_ymax,clp\_ymax clipping

\* Calculate the perspective view and draw it

bit\_loop:

bsr drw\_shw2

bsr perspective

bsr polydraw

bsr drw2\_shw1

pers\_loop

bra pers\_loop forever

\*SECTION DATA

include data\_01.s

include data\_02.s

\*SECTION BSS

include bss\_02.s

END

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Core\_02.s

\* Perspective stuff

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

include core\_01.s

perspective

move.w vncoords,d7 any points to do?

beq prs\_end

subq.w #1,d7 counter

lea vcoordsx,a0

lea vcoordsy,a1

lea vcoordsz,a2

lea scoordsx,a4

lea scoordsy,a5

link a6,#-32 open 16 word frame

prs\_crd

moveq #3,d6

lea persmatx,a3

prs\_elmnt

move.w (a0),d0

move.w (a1),d1

move.w (a2),d2

muls (a3)+,d0

muls (a3)+,d1

muls (a3)+,d2

add.l d1,d0

add.l d2,d0

move.w #1,d1

muls (a3)+,d1

add.l d1,d0

move.l d0,-(a6)

dbra d6,prs\_elmnt

move.l (a6)+,d3

bne prs\_ok

addq #1,d3

prs\_ok

addq.l #4,a6

move.l (a6)+,d4

divs d3,d4

add.w #160,d4

move.w d4,(a4)+

move.l (a6)+,d4

divs d3,d4

sub.w #199,d4

neg.w d4

move.w d4,(a5)+

addq.l #2,a0

addq.l #2,a1

addq.l #2,a2

dbra d7,prs\_crd

unlk a6

prs\_end

rts

polydraw

move.w npoly,d7

beq polydraw5

subq #1,d7

lea scoordsx,a0

lea scoordsy,a1

lea sedglst,a2

lea snedges,a3

lea col\_lst,a4

polydraw2

move.w (a4)+,d0

cmp.w #$1f,d0

ble polydraw3

move.w (a3)+,d0

addq.w #1,d0

add d0,d0

adda.w d0,a2

bra polydraw4

polydraw3

move.w d0,colour

move.w (a3)+,d0

beq polydraw3

move.w d0,no\_in

lea crds\_in,a5

polydraw1

move.w (a2)+,d1

lsl #1,d1

move.w 0(a0,d1.w),(a5)+

move.w 0(a1,d1.w),(a5)+

dbra d0,polydraw1

movem.l d7/a0-a4,-(sp)

bsr clip

bsr poly\_fill

movem.l (sp)+,d7/a0-a4

polydraw4

dbra d7,polydraw2

polydraw5

rts

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* BSS\_02.s

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

include bss\_01.s

scoordsx ds.w 100 xcoords

scoordsy ds.w 100 ycoords

sedglst ds.w 100 edge connections

snedges ds.w 20 number of edges in each polygon

npoly ds.w 1 number of polygons in this object

col\_lst ds.w 20 colours

vcoordsx ds.w 100 viewframe xcoords

vcoordsy ds.w 100

vcoordsz ds.w 100

vncoords ds.w 1

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Data\_01.s

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

include data.s

IFND TRANSFORM

my\_datax dc.w 115,115,25,25,43,107,43,40,65,65,40,75

dc.w 88,75,34,34,34,34,43,43

my\_datay dc.w -100,100,100,-100,-70,20,73,-55,-20,30

dc.w 53,-8,10,22,-40,-91,90,20,-50,48

my\_dataz dc.w 120,120,0,0,24,108,24,20,53,53

dc.w 20,66,84,66,12,12,12,12,24,24

ENDC

my\_edglst dc.w 0,1,2,3,0,4,5,6,4,7,8,9,10,7

dc.w 11,12,13,11,14,15,4,18,14,16,17,19,6,16

my\_nedges dc.w 4,3,4,3,4,4

my\_npoly dc.w 6

my\_colour dc.w 5,23,5,5,23,23

my\_xmin dc.w 0

my\_xmax dc.w 319

my\_ymin dc.w 0

my\_ymax dc.w 199

persmatx dc.w 100,0,0,0,0,100,0,0,0,0,0,0,0,0,1,100

\* data\_02.s

persmatx:

dc.w 100,0,0,0,0,100,0,0,0,0,0,0,0,0,,1,100

include equates.s

6. Basit Rotasyonlar

Burada yapmak istediğimiz şey, bir nesneyi dünya çerçevesinde döndürmektir. Dünya modelimizde bu, bir nesne nesne çerçevesinden dünya çerçevesine taşındığında olanların bir parçasıdır. Ek olarak, genel olarak, mevcut konumuna taşınırken ilişkili bir çeviri olacaktır. Eylemdeki basit döndürmelere bir örnek olarak, nesneden dünyaya dönüşüm, yapılacak iyi bir şeydir. Birkaç farklı nesnenin olduğu karmaşık bir dünyada, her birinin dünya resmini oluşturmak için hepsini bir araya getirmek üzere farklı çevirileri ve dönüşleri olurdu.

Başlamak için sadece bir nesnenin olduğu basit bir dünyayı ele alalım. Üzerinde çalışmak için zaten iyi bir örneğimiz var - perspektif dönüşümünü göstermek için kullanılan, üzerinde "A" yazan monolit. Veriler zaten girildi ve kullanıma hazır. Görmek istediğimiz şey monolitin ekranın ortasında dönmesi. Sırada bunu yapacağız.

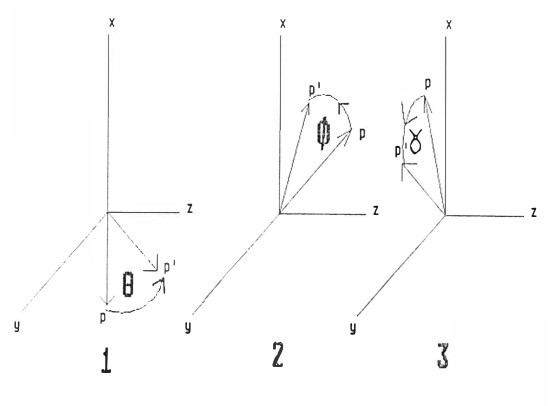
6.1. Geometrik Dönüşümler

Geometrik dönüşümler nesnelerin koordinatlarını değiştirenlerdir. Başka türleri var mı? Evet, referans çerçevelerini değiştiren, koordinat dönüşümleri olarak adlandırılan dönüşümler. Matematik dilinde geometrik dönüşüm, koordinat dönüşümünün tersidir (bu konu Ek 6'da da ele alınmıştır). İkinci türün bir örneği, dünya çerçevesinden görüntü çerçevesine dönüşümdür. Unutmayın, görüntü çerçevesi, dünya çerçevesinde hareket eden gözlemciye (siz) bağlı eksenler kümesidir. Hareket halindeki bir gözlemcinin görüntü çerçevesinden bakıldığında, tüm nesnelerin koordinatları sürekli olarak değişmektedir. Koordinat ve geometrik dönüşümler aynı madalyonun iki yüzü olmasına rağmen, görüntü dönüşümünü takip etmek biraz daha zordur ve daha sonra 9. ve 10. Bölümlerde yapılır. Bu bölümde x, y ve z eksenleri etrafındaki basit dönüşler matematiksel türetme olmadan sunulmaktadır. Ek bir matematiksel açıklama için Ek 6'ya bakın.

6.2. Ana Eksenler Etrafında Dönmeler

Bir topaç, dikey eksen etrafında geometrik dönüş geçiren bir nesnenin iyi bir örneğidir. Burada bizim ilgilendiğimiz kadarıyla, bunu yapmak için kullanılan matematik sadece 'ağır makine'dir. Bunu kullanmak için nasıl türetildiğini bilmeye gerçekten gerek yoktur. Tartışacağımız dönüşümler Şekil 6.1'de gösterilmiştir.

6.2.1. X ekseni etrafında dönüş



*Şekil 6.1: x, y ve z eksenleri etrafındaki dönüşler*

Bu, Şekil 6.1(1)'de koordinatları (x,y,z) olan bir P noktasının x ekseni etrafında 9 açıyla döndürülerek koordinatları (x' ,y' ,z') olan P' noktasına ulaşmasıyla gösterilmiştir. Noktaları vektörlerle temsil etmek dönüşü açıkça gösterir. Dönüş yönünün nasıl tanımlandığına dikkat edin. Yz düzleminin arkasından pozitif x ekseni boyunca bakıldığında saat yönündedir. Sütun vektörleri açısından, dönüşüm bir matris çarpımı olarak yazılabilir

x' 1 0 0 x

y' = 0 cos0 -sin0 \* y

z' 0 sin0 cos0 z

Basit cebirde, matris çarpımı çarpıldığında:

x' = x

y' = y.cos0 - z.sin0

z' = y.sinG + z.cosG

Kısaltılmış olması açısından matris R'(0) olarak kısaltılır ve dönüşüm daha sonra şu şekilde kısaltılır:

P' = R'(0).P

6.2.2. Y ekseni etrafında dönüş

Bu durumda P noktası, Şekil 6.1(2)'de gösterildiği gibi y ekseni etrafında 0 açısıyla döndürülür. Daha önce olduğu gibi, R'(0) dönüşü, xz düzleminin arkasından pozitif y ekseni boyunca bakıldığında saat yönündedir. Bir matris ürünü olarak ifade edildiğinde, dönüşüm şu şekildedir:

x' cos0 0 sin0 x

y' = 0 1 0 y

z' -sin0 0 cos0 z

6.2.3. Z ekseni etrafında dönüş

x' rahat siny 0 x

y' = siny rahat 0 i iy

z' 0 0 1 z

Şekil 6.1(3)'te P noktası z ekseni etrafında y açısıyla döndürülür. R' (y) dönüşü, xy düzleminin arkasından z ekseni boyunca bakıldığında saat yönündedir.

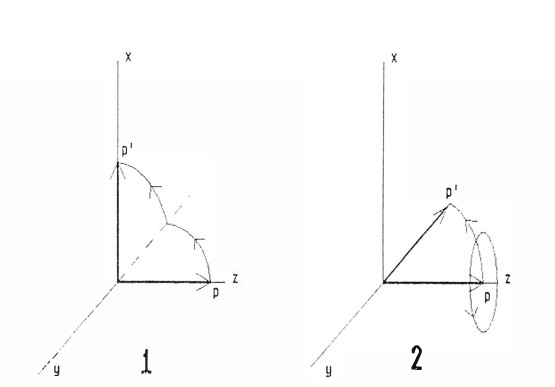
x' rahat -siny 0 x

y' = siny rahat 0 y

x' 0 0 1z

6.2.4. Bileşik Dönmeler

Tüm üç tür dönüş aynı anda yapıldığında işler çok daha karmaşık hale gelir. Bunun nedeni, dönüş sırasının önemli olmasıdır; ilk önce 9, ikinci <)> ve üçüncüyü y ile döndürmek, P'yi diğer herhangi bir sırada olduğu gibi aynı yerde sonlandırmaz. Bu şaşırtıcı bir sonuç gibi görünebilir. Matematiksel jargonla, üç boyutlu dönüşlerin değişmeli olmadığı söylenir. Bu noktayı açıklamak için Şekil 6.2'ye bakın.



*Şekil 6.2: Dönüşlerin sırası önemlidir*

Bunun iki kısmı vardır. 1. kısımda, başlangıçta z ekseninde bulunan bir vektör önce x ekseni etrafında 90° ve sonra z ekseni etrafında 90° döndürülür. Sonunda x ekseni boyunca işaret eder. 2. kısımda, dönüşlerin sırası tersine çevrilir. Sonuç olarak, ilk dönüş hiçbir şey yapmaz ve ikinci dönüş onu -y ekseni boyunca işaret eder halde bırakır. Açıkça, dönüş sırasını değiştirmek nihai sonucu değiştirir.

Bunun bir sonucu olarak, 0, <J> ve y'nin ayrı ayrı sayılması, son pozisyona ulaşmak için yeterli bilgi sağlamaz. Dönüş sırası da verilmelidir. Bireysel dönüşler küçük ve sık olduğunda, örneğin karmaşık bir yolu izleyen bir nesnede, yönelimi takip etmek için farklı bir strateji bulunmalıdır. Bu, 10. Bölüm'de tartışılmaktadır.

Şu an için bu pek de sorun değil. Dünya çerçevesinde veya nesneden dünyaya dönüşümün bir parçası olarak basit bir dönüş dizisi gerçekleştirmek, basit bir sırayla bireysel eksenler etrafında yalnızca üç dönüş gerektirebilir. Tutarlı bir şemaya sahip olmak için, önce y ile, ikinci olarak <J> ile ve üçüncü olarak 0 ile döndürürüz. Kısaca, tüm bu dönüşler bu sırayla gerçekleştiğinde genel dönüşüm şudur:

P' = R'(0).R'(0).R'(y).P

İlk dönüşün orijinal nokta P'nin yanında nasıl göründüğüne ve sonraki dönüşlerin daha sola doğru nasıl göründüğüne dikkat edin. Bu, sütun vektörleriyle matris çarpımının sırasıdır.

Vektör üzerinde matris ürünlerini ayrı ayrı gerçekleştirmeye gerek yoktur. Ürünleri önceden bulunarak tek bir sonuç matrisi elde edilebilir ve bu matris tek bir işlemde vektörle çarpılabilir. Bu birleşik (birleştirilmiş) dönüş R' (0,0,Y) ile gösterilir.

6.3. Nesneden Dünyaya Dönüşüm

Bu, konuştuklarımızı açıklamak için iyi bir dönüşüm.

Bu dönüşümün amacı, bir nesneyi nesne referans çerçevesinden, dünya resmini oluşturan diğer tüm nesnelerin kümesinde göründüğü dünya çerçevesine taşımaktır. Nesneden dünyaya dönüşüm, Şekil 6.3'te üç dönüşün ve bir ötelemenin genel durumu için gösterilmiştir. Bu durumda açılar dönüşüme özgüdür ve 00, o<J) ve oy olarak adlandırılır, böylece daha sonra diğer dönüşümlerde görünecek diğer açılardan ayırt edilirler ve yer değiştirme (Oox, Ooy, Ooz) veya vektör gösteriminde yazılır:

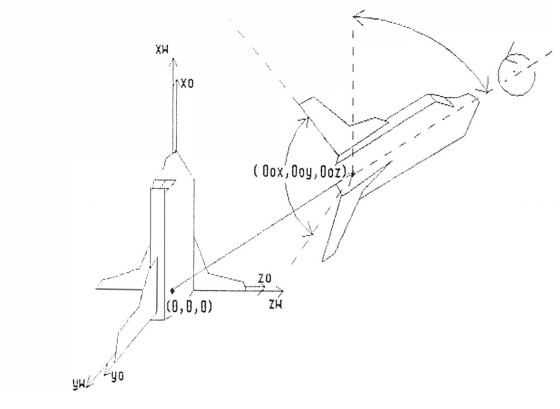
x' x Oox

y' = R' y \* Ooy

x'z Ooz

Çevirinin bir matris çarpımı olarak uygulanmadığını, ancak bir vektör eklemesi olarak bırakıldığını fark edin. Perspektif dönüşümü gibi, çeviri de onu her şeyle aynı zemine koymak ve birleştirmeye dahil edilmesine izin vermek için homojen koordinatlarda bir matris ürününe dönüştürülebilir. Bu burada yapılmamıştır çünkü basitçe dönüş dönüşümünü izleyen bir ekleme olarak dahil edilebilir. Homojen koordinatlar hakkında daha fazla bilgi Ek 6'da verilmiştir.

Nesne çerçevesini düşünmenin bir yolu, onu dünya çerçevesi kökenine merkezlenmiş bir eksen kümesi olarak düşünmektir. Bu kesinlikle geçerli bir resimdir çünkü herhangi bir dönüş veya öteleme olmadan nesne dünya çerçevesi kökeninde görünecektir. Öteleme, tüm nesnelerin dünya çerçevesi kökeninde üst üste gelmesini önlemek için önemlidir. Açılar çerçeveler arasında sürekli olarak değiştirilirse nesne dünya çerçevesinde dönecektir. Önceki bölümden perspektif dönüşümü zaten yerinde olduğundan bunun olmasını izleyebiliriz.



*Şekil 6.3: Dünya çerçevesinde genel geometrik dönüşüm*

6.4. Örnek Program

Bu, nesneden dünyaya dönüşümü kurmak ve bunu dünya çerçevesinin z ekseni etrafında dönen A monolitini göstermek için kullanmak için bir programdır. Ayrıca, dönüş matrisleri için açıların sinüsleri ve kosinüsleri hesaplanmalıdır. Bunların nasıl yapıldığı aşağıdaki örnek programlarda tartışılmaktadır.

6.4.1. otranw.s

Bu ana kontrol programıdır. Bu sefer başlatma daha kapsamlıdır çünkü çok fazla veri transferi gerçekleşir. A monolitini çizmek için gereken veri daha önce olduğu gibi data\_01.s dosyasındadır ancak şimdi nesne değişkenleri listesine aktarılması gerekir. Dönme, nesne çerçevesinden dünya çerçevesine aktarılırken gerçekleşir.

Şu anda yalnızca xw ekseni etrafında oQ açısıyla dönüşü gösterebiliyoruz. Bunun nedeni, diğer eksenler etrafında o<|) ve oy dönüşlerinin monolitin arka tarafını görüntülemeye çalışmasıdır. Bu, poligon doldurma rutininin ekran çerçevesindeki poligonların saat yönünün tersine bağlı bir kenar listesine sahip olmasını bekleyecek şekilde ayarlanması nedeniyle yapılamaz. Arka tarafta bu sıra tersine çevrilmiştir ve rutin bununla başa çıkmaya çalışırken çöp çizer. Normalde bir nesnenin arka tarafı görünmez ve bu şekilde ele alınır. Henüz görünürlüğü test etme olanağımız yok. Bu, Bölüm 7'de yapılır. Monolitin arkasını göstermek istenseydi, sırt sırta bir düzenlemede ayrı bir nesne olarak verilere girilmesi gerekirdi.

Program, A monolitinin dünya karesindeki zw ekseni etrafındaki dönüşünü 0° ila 360° açı aralığında 10° adımlarla gösterir. Her kare arasındaki açısal artışı ve yer değiştirmeyi (Oox, Ooy, Ooz) değiştirerek bunların ne gibi bir etki yarattığını görebilirsiniz. Çok büyük nesneler için, büyük bir nesnenin yalnızca küçük bir kısmının çizilmesini sağlayacak şekilde küçük bir pencereye sahip olmak iyi bir fikirdir, böylece boyut izlenimi kaybedilmeden hız korunur. Bu, birçok oyunda, her karede yeniden çizilmesi gereken tek parça olan çok küçük bir pencerenin ve başlangıçta yalnızca bir kez çizilen büyük bir statik kontrol panelinin olmasının nedenini açıklar.

6.4.2. data\_03.s

Dönme dönüşümü 00, cx)> ve <ry açılarının sinüs ve kosinüslerini kullanır. Basic'te çalışan bir program için bunlar bir seri yaklaşımı kullanılarak birçok anlamlı basamağa kadar hesaplanırdı. Burada buna zaman yok. El hesap makineleri icat edilmeden önce kullanılan yönteme başvurmalıyız - tablolar. Bu dosyadaki tablo, 0° ile 90° arasındaki tüm açıların sinüslerini, her biri 16384 faktörüyle, yani 214 ile çarpılmış 1° artışlarla içerir. Bunun nedeni basittir. Ondalık noktayı ikili sistemde 14 basamak sola kaydırır ve ürünlerin yüksek doğrulukta belirlenebilmesi için 1/16384 birimlerinde çalışmamızı sağlar. Ancak yeni bir koordinatın hesaplanmasının sonunda sonucun doğru boyutuna geri döndürülmesi için 16384'e bölünmesi gerektiği unutulmamalıdır. Ekranda görüntülenebilecek en küçük artış olduğundan, son koordinatı artı veya eksi 1'den daha büyük bir doğrulukla bilmenin bir anlamı yoktur. Ayrıca, tüm trigonometrik fonksiyonlar 16384 ile çarpılmamış olsaydı, tüm ürünler 0 ila 1 aralığına düşerdi ve ikili yaklaşımda bu değerlerden birine veya diğerine yaklaşırdı ve bu da tüm ürünler için sıfır veya aynı sonucu verirdi. 214'ü bir faktör olarak seçmenin amacı, 14 sola veya sağa kaydırma ile çok hızlı bir şekilde eklenip çıkarılabilmesidir. Daha büyük bir faktör kullanılarak daha büyük bir doğruluk elde edilebilirdi, ancak 16384, hataların oluştuğu yerde düzeltilmesi için adımlar atıldığı takdirde amaçlarımız için oldukça yeterlidir.

Daha yüksek hız için hem sinüsler hem de kosinüsler için ayrı tablolar olması daha mantıklıdır. Burada esas olarak sinüs ve kosinüslerin simetrisinin 0° ila 360° derece aralığındaki herhangi bir değerin 0° ila 90° derece aralığında hesaplanmasına nasıl izin verdiğini göstermek için yapılmamıştır. Bunu yapmak için gereken zaman, örneğin poligonu gerçekten doldurmak için gereken zamana kıyasla çok küçüktür, ancak daha yüksek hız için ayrı tablolar kullanılmalıdır.

6.4.3. çekirdek\_03.s

Buradaki alt programın ilk kısmı, dönüşüm matrisinde kullanılmaya hazır, dönüşte kullanılan açıların sinüslerini ve kosinüslerini bulmak için data\_03.s'deki arama tablosunu kullanır. Bu, 0° ila 360° aralığındaki herhangi bir açının sinüsünün veya kosinüsünün 0° ila 90° aralığındaki eşdeğer bir açıdan bulunabileceği sonucunu kullanır. Bu eşdeğer açıyı bulmak, ilk kısmın başlangıcının konusu.

İkinci bölümde, matris oluşturulur ve daha sonra nesne koordinatlarını matris çarpımı ile dönüştürmek için kullanılır, tıpkı daha önceki perspektif dönüşümünde yapıldığı gibi. Bu örnekte yalnızca x ekseni etrafında dönüşler yapılsa da, matris yukarıda açıklandığı gibi üç eksen etrafında da dönüşleri işleyebilir. Dönme dönüşümünün sonunda, nesneyi dünya çerçevesinde istenen konuma yerleştirmek için Oox, Ooy ve Ooz yer değiştirmeleri eklenir.

6.4.4. bss\_03.s

Yeni değişken listeleri.

\* otranw.s

\* Simple rotations for Chapter 6

\*

\*SECTION TEXT

opt d+

bra main

include systm\_00.s

include core\_03.s important subroutines

main bsr set\_up allocate memory etc.

\* transfer all the data

move.w my\_npoly,d7 no. of polygons

move.w d7,npoly pass it

subq.w #1,d7 the counter

move.w d7,d0 save it

lea my\_nedges,a0 source

lea snedges,a1 destination

lea my\_colour,a2 source

lea col\_list,a3 destination

loop0 move.w (a0)+,(a1)+ transfer edge nos.

move.w (a2)+,(a3)+ transfer colours

dbra d0,loop0

\* Calculate the number of vertices altogheter

move.w d7,d0 restore count

lea my\_nedges,a6

clr d1

clr d2

loop1 add.w (a6),d1 no more than this

add.w (a6)+,d2 total number of vertices

addq #1,d2 and last one repeated each time

dbra d0,loop1

\* Move the edge list

subq #1,d2 the counter

lea my\_edglst,a0 source

lea sedglst,a1 destination

loop2 move.w (a0)+,(a1)+ pass it

dbra d2,loop2

\* and the coords list

move.w d1,oncoords

subq #1,d1 the counter

lea oocoordsx,a1

lea my\_datax,a0

lea occoordsy,a2

lea my\_datay,a2

lea ocoords,aa5

lea my\_dataz,a4

loop3

move.w (a0)+,(a1)+

move.w (a2)+,(a3)+

move.w (a4)+,(a5)+

dbra d1,loop3

\* and the window limits

move.w my\_xmin,clp\_xmin ready

move.w my\_xmax,clp\_xmax for

move.w my\_ymin,clp\_ymin clipping

move.w my\_ymax,clp\_ymax

\* place it in the world frame

move.w #300,Oox 0 in the air

move.w #200,Ooz 100 in front

clr.w Ooy dead centre

\* initialise for rotation

clr.w otheta init angles

move.w #50,ophi tilt it up 50 degress

clr.w ogamma

clr.w screenflag 0=screen 1 draw, 1=screen 2 draw

\* Start the rotation about zw axis (can't rotate about others

\* or we'll see back of it).

loop5 move.w #360,d7 a cycle

loop4

move.w d7,ogamma next angle gamma

move.w d7,-(sp) save the angle

tst.w screenflag screen 1 or screen2?

beq screen\_1 draw on screen 1,display screen2

bsr drw2\_shw1 draw on screen 2, display screen1

clr.w screenflag and set the flag for next time

bra screen\_2

screen\_1:

bsr drw\_shw2 draw on 1, display 2

move.w #1,screenflag and set the flag for next time

screen\_2:

bsr otranw rotational transfers

\* pass on the new coords

move.w oncoords,d7

move.w d7,vncoords

subq.w #1,d7

lea wcoordsx,a0

lea wcoordsy,a1

lea wcoordsz,a2

lea vcoordsx,a3

lea vcoordsy,a4

lea vcoordsx,a5

loop6 move.w (a0)+,(a3)+

move.w (a1)+,(a4)+

move.w (a2)+,(a5)+

dbra d7,loop6

\* Complete the picture

bsr perspective perspective

bsr polydraw finish the picture

move.w (sp)+,d7

sub.w #10,d7 reduce the angle by 10 degrees

bgt loop4 next angle

bra loop5 or repeat the cycle

bra main this could go on forever

\*SECTION DATA

include data\_01.s

include data\_03.s

\*SECTION BSS

include bss\_03.s

END

\* data\_03.s

\* A sine look-up table

\*

\* table of sines from 0 to 90 degress in increments of 1 degree

\* multiplied by 2^14 (16384). Used to find the sine or cosine

\* of any angle

sintable:

dc.w 0,286,572,857,1143,1428,1713,1997,2280,2563,2845,3126

dc.w 3406,3686,3964,4240,4516,4790,5063,5334,5604,5872,613

dc.w 6402,6664,6924,7182,7438,7692,7943,8192,8438,8682,892

dc.w 9162,9397,9639,9860,10087,10311,10531,10749,10963,111

dc.w 11381,11585,11786,11982,12176,12365,12551,12733,12911

dc.w 13085,13255,13421,13583,13741,13894,14044,14189,14330

dc.w 14466,14598,14726,14849,14968,15082,15191,15296,15396

dc.w 15491,15582,15688,15749,15826,15897,15964,16026,16083

dc.w 16135,16182,16225,16262,16294,16322,16344,16362,16374

dc.w 16382,16384

include data\_02.s the perspective transfers

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Core\_03.s (subroutines for chapter six). \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* sincos - returns the sine and cosine of given angle

\* otranw - transforms obj coords to world coords.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

include core\_02.s

\* The sine and cosine of an angle are found. The sintable covers the positive quadrant \*

\* 0-90 degrees and can be used to generate any sin or cos in the range 0 - 360 degrees \*

\* d1=angle in degrees. Returns sin in d2; cos in d3.

sincos

lea sintable,a5

cmp #360,d1 test(angle-360)

bmi less360

sub #360,d1 make it less than 360

less360

cmp #270,d1 test(angle-270)

bmi less270

bsr over270

rts

less270

cmp #180,d1 test(angle-180)

bmi less180

bsr over180

rts

less180

cmp #90,d1

bmi less90

bsr over90

rts

less90

add d1,d1 \*2 for offset into table

move.w 0(a5,d1.w),d2 get sine

subi #180,d1 cos(angle)=sin(90-angle)

neg d1 offset into table for cosine

move.w 0(a5,d1.w),d3 cosine

rts

over270

subi #360,d1

neg d1 360-angle

add d1,d1 table offset

move.w 0(a5,d1.w),d2 get sine

neg d2

subi #180,d1 cos(angle)=sin(90-angle)

neg d1 offset into table for cosine

move.w 0(a5,d1.w),d3 cosine

rts

over180

subi #180,d1

add d1,d1 table offset

move.w 0(a5,d1.w),d2 get sine

neg d2

subi #180,d1 cos(angle)=sin(90-angle)

neg d1 offset into table for cosine

move.w 0(a5,d1.w),d3 cosine

neg d3

rts

over90

subi #180,d1

neg d1 360-angle

add d1,d1 table offset

move.w 0(a5,d1.w),d2 get sine

subi #180,d1 cos(angle)=sin(90-angle)

neg d1 offset into table for cosine

move.w 0(a5,d1.w),d3 cosine

neg d3

rts

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* The subroutines for transforming object coords to to world coords. \*

\* Includes rotations given by otheta, ophi and ogamma about the world axes wx,wy,wz and \*

\* a displacement Oox, Ooy, Ooz relative to the world origin. \*

\* Part 1. Construct the matrix for the rotations. \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Convert object rotation angles and store for rotation matrix.

otranw

move.w otheta,d1

bsr sincos

move.w d2,stheta

move.w d3,ctheta

move.w ophi,d1

bsr sincos

move.w d2,sphi

move.w d3,cphi

move.w ogamma,d1

bsr sincos

move.w d2,sgamma

move.w d3,cgamma

\* construct transform matrix otranw. (all elements end up doubled)

lea stheta,a0

lea ctheta,a1

lea sphi,a2

lea cphi,a3

lea sgamma,a4

lea cgamma,a5

lea o\_wmatx,a6 matrix

\* do element OM11

move.w (a3),d0 cphi

muls (a5),d0 cphi\*cgamma

lsl.l #2,d0

swap d0 /2^14

move.w d0,(a6)+ OM11

\* do OM12

move.w (a3),d0 cphi

muls (a4),d0 cphi\*sgamma

neg.l d0

lsl.l #2,d0

swap d0 /2^14

move.w d0,(a6)+ OM12

\* do OM13

move.w (a2),(a6)+ sphi

\* do OM21

move.w (a1),d0 ctheta

muls (a4),d0 ctheta\*sgamma

move.w (a0),d1 stheta

muls (a2),d1 stheta\*sphi

lsl.l #2,d1

swap d1

muls (a5),d1 stheta\*sphi\*cgamma

add.l d1,d0 stheta\*sphi\*cgamma + ctheta\*sgamma

lsl.l #2,d0

swap d0

move.w d0,(a6)+

\* do OM22

move.w (a1),d0 ctheta

muls (a5),d0 ctheta\*cgamma

move.w (a0),d1 stheta

muls (a2),d1 stheta\*sphi

lsl.l #2,d1

swap d1

muls (a4),d1 stheta\*sphi\*sgamma

sub.l d1,d0 ctheta\*cgamma - stheta\*sphi\*sgamma

lsl.l #2,d0

swap d0

move.w d0,(a6)+

\* do OM23

move.w (a0),d0 stheta

muls (a3),d0 stheta \* cphi

neg.l d0

lsl.l #2,d0

swap d0

move.w d0,(a6)+

\* do OM31

move.w (a0),d0 stheta

muls (a4),d0 stheta\*sgamma

move.w (a1),d1 ctheta

muls (a2),d1 ctheta\*sphi

lsl.l #2,d1

swap d1

muls (a5),d1 ctheta\*sphi\*cgamma

sub.l d1,d0 stheta\*sgamma-ctheta\*sphi\*cgamma

lsl.l #2,d0

swap d0

move.w d0,(a6)+

\* do OM32

move.w (a0),d0 stheta

muls (a5),d0 stheta\*cgamma

move.w (a1),d1 ctheta

muls (a2),d1 ctheta\*sphi

lsl.l #2,d1

swap d1

muls (a4),d1 ctheta\*sphi\*sgamma

add.l d1,d0

lsl.l #2,d0

swap d0

move.w d0,(a6)+

\* do OM33

move.w (a1),d0 ctheta

muls (a3),d0 ctheta\*cphi

lsl.l #2,d0

swap d0

move.w d0,(a6)+

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* PART 2: transform object coords to world coords. matrix elements are 2^14 and must be \*

\* adjusted when we're finished.

move.w oncoords,d7 number

ext.l d7 any to do ?

beq otranw3

subq.w #1,d7 adjust counter for dbra

lea ocoordsx,a0

lea ocoordsy,a1

lea ocoordsz,a2

lea wcoordsx,a3

lea wcoordsy,a4

lea wcoordsz,a5

exg a3,d3 save address( not enough a regs!!)

link a6,#-6 stack frame of 3 words

otranw1

moveq.l #2,d6 3 rows in the matrix

lea o\_wmatx,a3 point at matrix

\* calculate the next wx,wy and wz

otranw2

move.w (a0),d0 ox

move.w (a1),d1 oy

move.w (a2),d2 oz

muls (a3)+,d0 ox\*MI1

muls (a3)+,d1 oy\*MI2

muls (a3)+,d2 oz\*MI3

add.l d1,d0

add.l d2,d0

lsl.l #2,d0

swap d0

move.w d0,-(a6) save it

dbra d6,otranw2 repeat for three elements

move.w (a6)+,d0

add.w Ooz,d0 add displacement

move.w d0,(a5)+ becomes wz

move.w (a6)+,d0

add.w Ooy,d0

move.w d0,(a4)+ becomes wy

exg a3,d3 restore wx, save matrix pointer

move.w (a6)+,d0

add.w Oox,d0

move.w d0,(a3)+ becomes wx

exg a3,d3 save wx restore matrix pointer

addq.l #2,a0 point to next ox

addq.l #2,a1 oy

addq.l #2,a2 oz

dbra d7,otranw1 repeat for all coords

unlk a6

otranw3

rts

\* bss\_03.s

\*

include bss\_02.s

\* Object frame variables

otheta ds.w 1 rotation of object coords about wx

ophi ds.w 1 ditto wy

ogamma ds.w 1 ditto wz

ocoordsx ds.w 200 vertex x coords

ocoordsy ds.w 200 ditto y

ocoordsz ds.w 200 ditto z

oncoords ds.w 1 number

Oox ds.w 1 object origina x in world frame

Ooy ds.w 1 ditto y

Oox ds.w 1 ditto z

\* World frame variables

wcoordsx ds.w 200

wcoordsy ds.w 200

wcoordsz ds.w 200

\* Variables for the o\_w transform

o\_wmatx ds.w 9 the matrix elements

\* General

screenflag ds.w 1 0 display screen 1, 1 for screen 2

stheta ds.w 1 trig functions of current angle

ctheta ds.w 1

sphi ds.w 1

cphi ds.w 1

sgamma ds.w 1

cgamma ds.w 1

7. Gizli Yüzeyler ve aydınlatma

Bilgisayar hızlı bir sayı işlemcisidir, ancak gerçek dünya hakkında hiçbir şey bilmez. Katı opak nesnelerin içini görememek gibi basit günlük deneyimleri iletmeye gelince, bilgisayar gerçek bir kaybedendir. İşlemci talimat setinde bu tür bilgileri kolayca iletmemizi sağlayan kodlar yoktur. Opak nesnelerin arka taraflarının görünmediği ve opak bir nesnenin arkasındakileri gizleyeceği bize aşikar görünüyor. Bilgisayarın bu basit yaşam gerçeğini göstermesini sağlamak zor bir iştir. Buna gizli yüzey problemi denir ve bilgisayar grafiklerindeki çok zaman alan bazı algoritmaların temelidir.

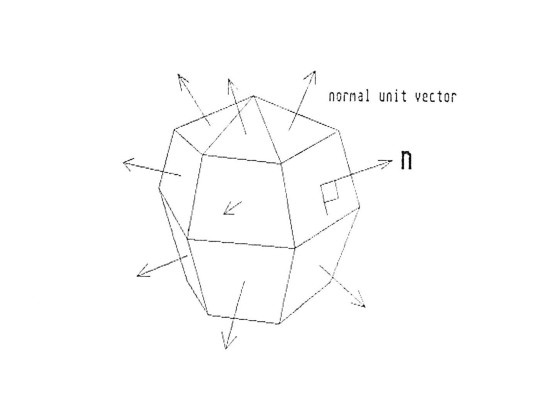
Adanmış grafik donanımı olmayan herhangi bir mikro için, hesaplama yükü ana işlemciye düştüğünden ve bu nedenle gizli yüzeylerle başa çıkmak için benimsediğimiz herhangi bir stratejinin çok zaman alıcı olmaması gerektiğinden, bu ciddi bir sorun haline gelir. Sonuç olarak, nesnelerin geometrisi, zaman alıcı bir gizli yüzey algoritması gerektirecek kadar karmaşık olamaz. En basit çözüm, tüm çokgenlerin dışbükey olmasını, yani her yüzey çokgeninin dışarıya bakmasını ve başka bir çokgene doğru bakmamasını gerektirmektir. Dışbükey olmayan basit çokgenlerle başa çıkmak mümkündür, ancak yalnızca dışbükey olanları ele alacağız. Birkaç dışbükey çokgenden karmaşık nesneler oluşturmak her zaman mümkündür ve bu durumda strateji önce en uzaktakileri ve en yakın olanları en sona çizmektir. Bu, arka plandaki nesnelerin ön plandakiler tarafından doğal olarak gizlendiği sözde 'ressam' algoritmasıdır. Bunlara daha sonra değineceğiz.

Bir yüzeyin görünür olup olmadığına karar verme prosedürü, nesnenin gerçek görünmesi için gerekli bir özellik olan uzak bir ışık kaynağı tarafından ne kadar parlak aydınlatıldığına karar verme hesaplamasıyla doğal olarak birleşir. Işık kaynağına doğru bakan yüzeyler, uzağa bakanlardan daha parlak olmalıdır. Bu ikisini bu bölümde tek bir algoritmada birleştireceğiz.

7.1. Gizli Yüzey Kaldırma

Burada benimsenen dışbükey çokyüzlüler için basit stratejide, bir yüzeyin görünür olup olmadığına karar vermek önemli miktarda vektör cebiri gerektirir (bu, belirli yüzey parametrelerini önceden hesaplayarak en aza indirilebilir). Prosedür basittir: çokgen bir yüzey, bakış açısına bakıyorsa görünürdür. Sorun, "yüzler" kelimesinin matematiksel bir ifadeye nasıl dönüştürüleceğidir. Bu, aşağıdaki şekilde yapılır.

Her yüzey, yüzeyden dik açılarda işaret eden bir vektörle ilişkilendirilmiştir, böylece çokyüzlü bir bütün olarak bir kirpi gibi görünür. Bu tür tüm vektörlerin aynı uzunluğu vardır ve bu uzunluk birlik olarak seçilir. Bunlara yüzey normal birim vektörleri denir. İki birim vektör arasındaki tek fark, Şekil 7.1'de gösterildiği gibi yüzeylerin baktığı farklı yönleri yansıtan yönleridir. Elbette, hesaplama amaçları için 1 bir vektör için kullanışlı bir boyut değildir ve bu nedenle 16384 (214) faktörüyle çarpılır. Bu, miktarları kelime boyutunda tutar ve çarpma ve bölmeyi basit hale getirir.



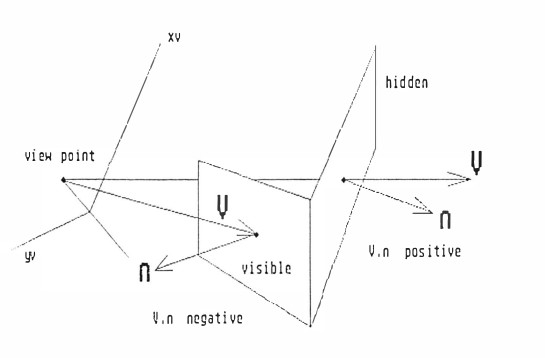
*Şekil 7.1 Yüzey normal vektörlerini gösteren dışbükey bir çokyüzlü*

Bir yüzeyin bakış açısından görünür olup olmadığını belirlemek artık birim vektörünün bakış açısından yüzeye çizilen bir vektöre (görünüm vektörü) aynı veya zıt yönde olup olmadığını test etmekten ibarettir. Bu testi gerçekleştiren temel bir vektör çarpımı vardır. Buna skaler veya nokta çarpımı denir. Ek 6 vektörleri içeren çarpımları açıklar. Matematik dilinde, görünüm vektörü V ve yüzey normal vektörü n olduğunda, skaler çarpım yüzey gizliyse pozitif sonuç, görünürse negatif sonuç verecektir:

gizli: skaler ürün Vn pozitiftir

görünür: skaler çarpım Vn negatiftir.

Skaler çarpım aslında bakış noktasından yüzeye olan mesafe ile bakış vektörü ile yüzey normali arasındaki açının kosinüsünün çarpımından başka bir şey değildir. Bu nedenle, çarpımın işareti doğal olarak 90°'den küçük bir açının kosinüsünün pozitif, 90° ile 180° arasındaki bir açının kosinüsünün ise negatif olmasından kaynaklanır. Şekil 7.2, görünür ve gizli bir yüzey için vektörlerin yönlerini göstermektedir. Bunların hepsi çok tatmin edicidir, tek bir şey hariç; yüzey normali birim vektörü hesaplanmalıdır ve bu o kadar da basit değildir. Burada birim vektör, bakış çerçevesi koordinatlarında hesaplanır.



*Şekil 7.2 Görünür ve gizli yüzeyler*

Kısa bir konu dışı olarak, görünürlük testinin vektör ürünlerine herhangi bir referans olmadan yapılabileceğini belirtmekte fayda var. Veri listelerinin, önden bakıldığında bir poligonun kenar bağlantıları listesinin saat yönünde gittiği şekilde ayarlanma şekli, görünürlük için basit bir test vermek için kullanılabilir. Perspektif dönüşümü ile ekran koordinatlarına dönüştürüldüğünde, görünür poligonların kenar listesi saat yönünün tersine gider. Bu nedenle, saat yönünde ekran kenar listelerine sahip yansıtılmış poligonlar, ekrandan uzağa bakan ve gizlenmesi gereken poligonlardan gelmiş olacaktır. Bunun için bir test kolayca oluşturulabilir.

Burada skaler çarpımı kullanmayı tercih etmemizin nedeni, normal birim vektörlerin bir kez hesaplandıktan sonra her bir yüzeyin aydınlatma seviyesini belirlemek için de kullanılabilmesidir.

7.2. Yüzey Normal Birim Vektörünün Hesaplanması

Normal birim vektörlerini hesaplama prosedürü oldukça fazla vektör cebiri ve zaman alıcı çarpmalar gerektirir. Önceden bazı ilgili nicelikleri hesaplayarak ve verileri her zamanki gibi bir listede depolayarak en aza indirilebilir. Aslında normal vektörlerin kendileri nesne çerçevesinde tamamen hesaplanabilir ve her aşamada köşelerle birlikte dönüştürülebilir. Bunu bu şekilde yapmanın önemli avantajları vardır.

Bunun yerine, programımızın evrimine ve kitabın öğretici amacına güzelce uyması nedeniyle vektörleri görünüm çerçevesi koordinatlarında hesaplamayı tercih ediyoruz. Normal vektörü hesaplamamızı sağlayan belirli vektör çarpımına çapraz çarpım denir. Skaler çarpımdan anlaşılması daha zordur ancak tam olarak istediğimiz şeydir. Ek 5 de bu konuyu ele almaktadır.

Şekil 7.3'te tek bir çokgen için bir vektör ürünü gösterilmiştir. Çokgenin çevresini dolaşırken karşılaştığımız ilk iki kenar 1'den 2'ye ve 2'den 3'e kadar olan köşelerdir. Bu kenarlarla ilişkili vektörlere A12 ve A23 diyelim. Normal vektör B daha sonra bunlar arasındaki çarpım çarpımı olarak hesaplanır:

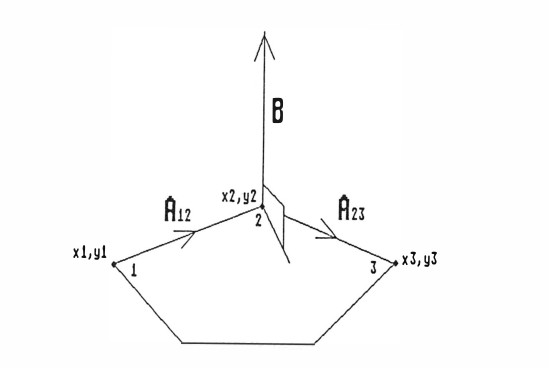
B = A23 x A12.

Bu kısaltma gösterimi, bir dizi matematiksel işleme çevrilene kadar oldukça anlamsızdır. A12 ve A23'ün x, y ve z bileşenleri şunlardır:

A12x = x2-x1, A12y = y2-y1, A12z = z2-z1 A23x = x3-x2, A23y = y3-y2, A23z = z3-z2

ve B'nin bileşenleri şunlardır:

Bx = A12z.A23y-A12y.A23z Bir = A12x.A23z-A12z.A23x



*Şekil 7.3 İki vektörün vektörel çarpımı*

Bu çarpımlar hesaplamanın büyük kısmını oluşturur.

Son bir adım daha var. İstediğimiz şey birim vektör. B vektörü doğru yönde ancak boyutu çok büyük. Birim vektörü elde etmek için, bileşenlerin her biri B'nin büyüklüğüne bölünmelidir. Bu, B'nin büyüklüğünün şu şekilde hesaplanması nedeniyle ek bir iş çıkarır:

B = V(Bx2+By2+Bz2)

karekök alınmasını gerektirir. Bunun nasıl yapıldığı örnek programda açıklanmıştır.

B büyüklüğü hesaplandıktan sonra, birim vektörün bileşenleri şunlardır:

bx = Bx/B, by = By/B, bz = Bz/B.

Bundan sonra, görüş noktasından kenar listesindeki yüzeyin ilk köşesine kadar olan görüş hattı vektörü (görünüm vektörü) bulunur ve normal vektörle skaler çarpım alınır. Bu teste dayanarak, yüzey ya gizli olarak işaretlenir ya da aydınlatma seviyesi hesaplanır. Aydınlatmayı daha sonra tartışacağız.

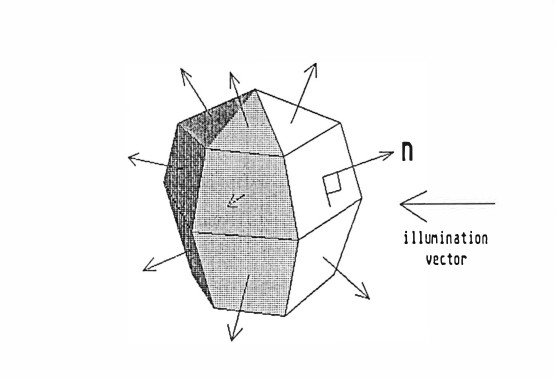
7.3. Aydınlatma ve Renk

Geometrik olarak doğru 3B modeller oluşturmak için en ayrıntılı hesaplamaları kullanmak mümkündür, ancak bunları gerçek gibi gösteren nitelikler çok incelikli ve daha az belirgin olabilir. Sprite grafiklerinde, bir merminin hareketini izleyen yerdeki gölge, yüksekliğine dair küçük ama önemli bir ipucudur. 3B'de, bir modele gerçekçilik katmak için en kolay ve çarpıcı iyileştirmelerden biri, bir ışık kaynağı tarafından aydınlatılmasıdır. Işık kaynağına bakan yüzeyler, uzağa bakan yüzeylere göre daha parlak bir şekilde aydınlatılır. Nesne yönünü değiştirdikçe, aydınlatmadaki değişiklikler şekli ve yapısı hakkında ek görsel ipuçları verir. Bundan sonra simüle etmeye çalışacağımız şey budur. Amiga'da elde edilebilecek şeylerde, yazılım kısıtlamalarının bir sonucu değil, daha çok rengin renk paletinde uygulanma biçiminden kaynaklanan sınırlamalar vardır. Aydınlatmanın belirlenme biçimi, görünürlüğün test edilme biçimine çok benzer, ancak bu durumda yüzeyin ışık kaynağına olan açısına bağlı olarak gerçek bir sayı üretilmelidir.

Bir ışık kaynağından çıkan ışık huzmesinin yönü, aydınlatma vektörü adı verilen bir vektörle belirtilir. Bu vektörün aydınlatma alanı boyunca yönünü değiştirmesiyle ıraksayan veya yakınsayan bir huzmeyi simüle etmek mümkün olabilir, ancak basitlik açısından huzmenin paralel olduğu kabul edilir. Sonuç olarak, huzmenin yönünü tanımlamak için tek bir vektör yeterlidir. Aynı şekilde, ışığın yoğunluğu her yerde sabit kabul edilir. Bu yaklaşımlar Güneş gibi uzak bir ışık kaynağı için geçerlidir, ancak yakın bir ışık kaynağı için fark neredeyse fark edilmez. Bu aydınlatma vektörü de bir birim vektördür (yani büyüklüğü birdir.)

Yüzey normal birim vektörlerini zaten hesapladığımız için, her şey yüzeydeki her bir fasetin aydınlatma seviyesini bulmak için ayarlanmıştır. Şekil 7.4 hesaplamayı göstermektedir. Aydınlatma vektörü ile normal vektörlerin skaler çarpımından başka bir şey değildir. Aydınlatma seviyesi iki vektör arasındaki açının kosinüsüne bağlı olduğundan bu gerçekçi bir hesaplamadır.

Hesaplamada kullanacağımız küçük bir değişiklik var. Dünya'nın Güneş tarafından nasıl aydınlatıldığını düşünün: Güneş'e bakan taraf parlak bir şekilde aydınlatılmıştır ancak Ay'ın yansıyan ışığı olmasaydı (yıldızların ışığını unutarak) diğer taraf zifiri karanlık olurdu. Bir odada tek bir ışık kaynağı her şeyi aydınlatmak için yeterlidir, ancak bunun çoğu duvarlardan ve odadaki tüm nesnelerden yansıyan ışıktır. Bu, gerçekçiliği yüksek derecede simüle etmek için çok gelişmiş grafiklerde kullanılan aydınlatma hesaplaması için Radyosite yönteminin temelidir. Bunun çok ilkel bir versiyonunu yöntemimize dahil edebilir, negatif olduğu yerlerde bile bir aydınlatma seviyesi belirlemek için skaler ürünü kullanabiliriz, böylece nesnelerin karanlık tarafında bile bir miktar aydınlatma olur.

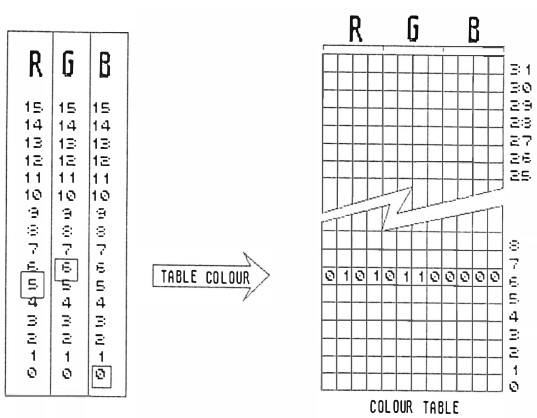


*Şekil 7.4 Yüzey aydınlatması*

İşte ana hatlarıyla yöntem: her yüzey için, aydınlatma vektörünün normal birim vektörle skaler çarpımını alın; her iki vektör de büyüklük olarak 1 olduğundan, bu +1 (minimum aydınlatma) ile -1 (maksimum aydınlatma) arasında bir sonuç verecektir. İşaret sizi karıştırdıysa, geometrimizde aydınlatma vektörünün ışık kaynağından uzağa baktığını unutmayın. Yöntemimizde tüm birim vektörler 214 (16384) ile çarpıldığı için, skaler çarpım aslında -228 ile +228 aralığında bir sonuç verecektir. Bu sonuca 2^ eklenip 224'e (sağa kaydırarak) bölündüğünde, bu 0 ile 32 aralığına düşer. Bu sonuç daha sonra 32 farklı renk tonunu indekslemek için kullanılabilir. Bunun nasıl yapıldığı, renk tablosunun tekrar kısa bir açıklamasını gerektirir.

7.3.1. Renk Tablosu

Düşük çözünürlükte, olası 4096'dan 32 farklı renk aynı anda görüntülenebilir. Bu 32'lik seçime renk tablosu veya paleti denir. Bir resim çizilirken (örneğin yatay boşluk sırasında) renk paletini sık sık değiştirerek ekranın tamamı için 32'yi aşmanın püf noktaları vardır. Temel 32'yi kullanacağız. Aşağıdakiler için Şekil 7.5 yardımcı olacaktır. Standart palet ayarları data\_01 .s dosyasında listelenmiştir.



*Şekil 7.5: Renk tablosu için renklerin oluşturulması*

Temel olarak, bir renk kırmızı, yeşil ve mavinin her biri on altı yoğunluktan birinde birleştirilerek yapılır. Bu, 16x16x16 = 4096 olası kombinasyon olduğu anlamına gelir. Herhangi bir anda bu 4096 renkten 32'si ekranda aynı anda gösterilebilir. Neden 32? Çünkü 3. Bölüm'de gördüğümüz gibi düşük çözünürlükte 5 renk düzlemi vardır ve her düzlem bir bit ile temsil edilir, böylece 32'ye kadar kombinasyon mevcuttur. Rengin 5 bitlik değeri, rengin kelime numarasını içeren renk paletinden bir 'pot'u dizinlemek için kullanılır.

Geriye kalan tek şey, paletteki renk kelimesinin kırmızı, yeşil ve mavi ayarlarından nasıl üretileceğini bulmaktır. Aslında renkler, onaltılık olarak yazıldığında tam olarak sunuldukları gibi takip eder. $0fff (while) ayarı, kırmızı = $f, yeşil = $f ve mavi = $f anlamına gelir. Bunları ondalık olarak yazmak istiyorsanız, tarif şudur:

renk değeri = 256\*(kırmızı ayar) + 16\*(yeşil ayar) + l\*(mavi ayar)

Seçilen renkler daha sonra palete yüklenmelidir. Örnek programda yapılan budur. Bizim amacımız için, aydınlatmayı simüle etmek için renkler aynı rengin farklı tonları olacaktır. Burada açıkça bir takas olacaktır. Maksimum 32 renkle, aşağıdaki kombinasyonlar burada dikkate alınır:

mod 0 32 bir rengin tonu

mod 1 2 rengin 16 tonu

mod 2 4 rengin 8 tonu

mod 3 8 rengin 4 tonu

Bizim kullanacağımız mod 2 olacak, diğerleri yazılımda mevcut.

7.4. Örnek Programlar

Örnek programlar, A monolitini gizli yüzey kaldırma ve aydınlatma ile dönerken gösterir. Program x ekseni etrafında dönerek ayarlanmıştır ancak bu istenildiği gibi değiştirilebilir. Monolit kırmızı ve mavi renktedir ancak aşağıda açıklandığı gibi içsel renklerini değiştirerek yeşil ve beyaza dönüştürülebilir. Yukarıdaki renk tarifini izleyerek farklı renklerde alternatif paletler ayarlamak da çok eğlencelidir.

7.4.1. illhide.s

Bu kontrol programıdır. A monolitinin verilerini, nesne çerçeve eksenlerinden herhangi biri veya üçü etrafında dönerken görüntülemek için kullanır. Artık gizli yüzey kaldırma işlemimiz olduğundan, açıların arka tarafı görüntüleyecek kadar büyük olması önemli değildir. Arka taraf gizli olduğu için hiçbir şey görüntülenmez. Program, nesne çerçevesinin x ekseni etrafında dönüş için ayarlanmıştır.

Renk paleti 7 mavi tonu ve 8 kırmızı tonu, 8 yeşil tonu ve 8 beyaz tonu kullanacak şekilde ayarlanmıştır. Paletteki ilk renk 0 değerine sahiptir, siyahtır ve sistem tarafından arka planı sağlamak için kullanılır. Gölgelendirme modu esnektir ve yukarıdaki mod numarasına eşit bir değere sahip illkey adlı bir anahtar aracılığıyla ayarlanır. Program mod 2'de ayarlanmıştır.

7.4.2. çekirdek\_04.s

Bu, yüzey normal vektörlerini hesaplar, bir yüzeyin görünür olup olmadığını belirler ve eğer öyleyse, metinde ana hatlarıyla belirtilen aydınlatma seviyesini ve son palet rengini hesaplar. Normal vektörlerin hesaplanmasında kelime çarpımının sınırlamaları nedeniyle, nesneler yaklaşık 200'den küçük doğrusal boyutlarla sınırlandırılmıştır.

Öncelikle yüzey normal vektörleri yukarıda açıklandığı gibi hesaplanır. nrm\_vec alt rutininde normal vektör, bileşenlerinin her biri vektörün büyüklüğüne bölünerek bir birim vektöre dönüştürülür. Büyüklük, 3B'de Pisagor teoremi ile hesaplanır ve yinelemeli bir işlemle sqrt alt rutininde yapılan bir karekök işlemi gerektirir.

Karekök algoritması şu şekilde çalışır. Bir sayının, N, karekökünün yaklaşık olarak bilindiğini varsayalım; buna sqrt1 diyelim. Daha sonra daha iyi bir yaklaşık değer olan sqrt2, sayıyı sqrt1'e bölerek, bunu sqrt1'e ekleyerek ve 2'ye bölerek bulunabilir, yani

sqrt2 = l/2(sqrt1 + N/sqrt1).

sqrt2, sqrt1'den daha iyi bir yaklaşımdır. Daha sonra sqrt2 ile başlanarak daha da iyi bir yaklaşım olan sqrt3, aynı şekilde bulunabilir. Bu yeniden hesaplamaların her birine yineleme denir. Mütevazı bir yaklaşıklıkla başlanarak, 216'da 1 parçaya kadar doğrulukta bir karekök hesaplamak için rutinde yalnızca üç yinelemeye ihtiyaç vardır, yani bir kelimenin izin verdiği kadar doğruluk.

Görüşte görünürlüğü belirlemek için kullanılan görüş hattı vektörü, bakış noktasından bir yüzeydeki ilk tepe noktasına kadar alınır. Burada hiçbir belirsizlik yoktur çünkü bir yüzeyin görünür olmaktan çıktığı noktada tüm tepeler yüzey normaline dik bir görüş hattı vektörü verir. Aydınlatma vektörü, her zamanki gibi doğruluk için her biri 214 ile çarpılan ill\_vecx, ill\_vecy ve ill\_vecz bileşenleri tarafından belirtilir.

Bir yüzey görünmezse, aydınlatma $lf değerine ayarlanır. Aksi takdirde, mod 2'deki (burada kullanılan mod) içsel renk, 0, 1, 2 veya 3, renk paletini dizinlemek için bir sayı üretmek üzere gölgelendirmeyle birleştirilir. Bu, zor bir hesaplamadır ve algoritmayı takip ederek en iyi şekilde anlaşılır.

Özellikle, rengin 0 veya 1 olduğu duruma bakalım, böylece 1 ila 7 (mavi) (0 siyah, arka plan için ayrılmıştır) veya 8 ila 15 (kırmızı) arasındaki tonlar seçilir. Gerçek gölgelendirme seviyesi daha sonra grupta hangi rengin seçileceğini sabitler, en açık olan 1 (mavi) ve 8 (kırmızı) ve en koyu olan 7 (mavi) ve 15 (kırmızı) olur.

7.4.3. veri\_04.s

Bu, bu örnekte görünüm çerçevesinde sağdan sola parlayan bir ışık kaynağı tanımlayan aydınlatma vektörü bileşenlerini içerir. Bu, genel olarak iyi değildir çünkü ışık kaynağı dünya çerçevesinde sabitlenmeli ve diğer her şey gibi görünüm çerçevesine dönüştürülmelidir.

Bunu, mod 2'deki 0, 1, 2 veya 3 olmak üzere dört olasılığa karşılık gelen içsel renkler (bu durumda mavi, kırmızı, yeşil veya gri) takip eder. Paletteki renkler yukarıda açıklandığı gibi onaltılık olarak listelenmiştir.

7.4.4.bss\_04.s

Ek değişkenler.

\* illhide.s

\* A program illustrating illumination and hidden surface removal

\*

\*SECTION TEXT

opt d+

bra main

include systm\_00.s

include core\_04.s illumination, hidden surface removal

main bsr init set up memory and new palette, etc

\* transfer all the data from my lists to progress lists

bsr transfer

\* place it in the world frame

move.w #0,Oox on the ground

move.w #100,Ooz 100 in front

clr.w Ooy dead centre

\* Initialise angles for rotation

clr.w otheta

move.w #50,ophi tilt it forward

clr.w ogamma

\* Initalize screens

clr.w screenflag 0=screen 1 draw, 1=screen 2 draw

\* Start the rotation about the xw axis

loop5 move.w #360,d7 a cycle

loop4 move.w d7,otheta next theta

move.w d7,-(sp) save the angle

tst.w screenflag screen 1 orscreen2?

beq screen\_1 draw on screen 1, display screen2

bsr drw2\_shw1 draw on screen 2, display screen1

clr.w screenflag and set the flag for next time

bra screen\_2

screen\_1:

bsr drw1\_shw2 draw on 1,display 2

move.w #1,screenflag and set the flag for next time

screen\_2:

bsr otranw object-to-world transform

\* pass on the new coords

move.w oncoords,d7

move.w d7,vncoords

subq.w #1,d7

lea wcoordsx,a0

lea wcoordsy,a1

lea wcoordsz,a2

lea vcoordsx,a3

lea vcoordsy,a4

lea vcoordsz,a5

loop6 move.w (a0)+,(a3)+

move.w (a1)+,(a4)+

move.w (a2)+,(a5)0

dbra d7,loop6

\* Test for visibility and lightning

bsr illuminate if it is viisble find the shade

\* Complete the drawing

bsr perspective perspective

bsr polydraw finish the picture

move.w (sp)+,d7

sub.w #10,d7 decrement in 10 degree stpes

bgt loop4

bra loop5

\* SECTION DATA

include data\_01.s

include data\_03.s

include data\_04.s

\* SECTION BSS

include bss\_04.s

END

m\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Core\_04.s

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

include core\_03.s

illuminate

calc\_nrm

move.w npoly,d7

beq nrm\_out

subq #1,d7 counter

lea vcoordsx,a0

lea vcoordsy,a1

lea vcoordsz,a2

lea sedglst,a3

lea snedges,a4

lea snormlst,a5

\* calculate the surface normal unit vectors

next\_nrm

move.l a5,-(sp) save pointer to normals list

move.w (a3),a5 first vertex of next surface

move.w 2(a3),a6 second vertex

add a5,a5 \*2 for offset

add a6,a6 again

move.w 0(a0,a6.w),d1 x2

sub.w 0(a0,a5.w),d1 x2-x1 = A12x

move.w 0(a1,a6.w),d2 y2

sub.w 0(a1,a5.w),d2 y2-y1 = A12y

move.w 0(a2,a6.w),d3 z2

sub.w 0(a2,a5.w),d3 z2-z1 = A12z

move a6,a5

move.w 4(a3),a6 third vertex

add a6,a6 \*2 for offset

move.w 0(a0,a6.w),d4 x3

sub.w 0(a0,a5.w),d4 x3-x2 = A23x

move.w 0(a1,a6.w),d5 y3

sub.w 0(a1,a5.w),d5 y3-y2 = A23y

move.w 0(a2,a6.w),d6 z3

sub.w 0(a2,a5.w),d6 z3-z2 = A23z

movea.w d2,a5 save

muls d6,d2

movea.w d3,a6 save

muls d5,d3 ditto

sub.l d2,d3 Bx

move.l d3,-(sp) save to stack

move.w a5,d2 restore

move.w a6,d3 restore

movea.w d3,a5 save

muls d4,d3

movea.w d1,a6

muls d6,d1

sub.l d3,d1 By

move.l d1,-(sp) save it

move.w a6,d1 restore

\* last component - no need to save values

muls d5,d1

muls d4,d2

sub.l d1,d2 Bz

move.l d2,-(sp) save it

movem.l (sp)+,d4-d6 Bx in d6, By in d5 and Bz in d4

nrm\_cmpt

lsr.l #2,d4 /4 to prevent overspill

lsr.l #2,d5

lsr.l #2,d6

move.w d4,d0

move.w d5,d1

move.w d6,d2

move.l d7,-(sp) save

bsr nrm\_vec calculate unit vectors bx, by, bz

move.l (sp)+,d7 restore

move.w d0,d4

move.w d1,d5

move.w d2,d6

move.l (sp)+,a5 retore pointer to normals list

move.w d6,(a5)+ save nx

move.w d5,(a5)+ save ny

move.w d4,(a5)+ save nz

move.w (a4)+,d0 num vertices in this surface

add #1,d0 edge list always repeats the first

add d0,d0 \*2 for offset

adda.w d0,a3 adjust pointer to next surface

dbra d7,next\_nrm do all surfaces

nrm\_out

vis\_ill

\* Find visibility and level of illumination of surface by taking the scalar

\* product of the surface normal vector with the line of sight vector from viewpoint

\* and illumination respectively.

move.w npoly,d7

subq.w #1,d7

lea vcoordsx,a0

lea vcoordsy,a1

lea vcoordsz,a2

lea sedglst,a4

lea snedges,a3

lea snormlst,a5

lea slumlst,a6

move.w ill\_vecx,d0

move.w ill\_vecy,d1

move.w ill\_vecz,d2

\* line of sight vector is taken between the first vertex on the surface and viewpoint

next\_ill

move.w (a4),d6 1st point on next surface

add d6,d6 for offset

move.w 0(a0,d6.w),d3 is line of sight x cmpnt, x1s

move.w 0(a1,d6.w),d4 yLs

move.w 0(a2,d6.w),d5 z

sub.w vwpointz,d5 zls: vpoint lies on -zv axis

muls (a5),d3 nx\*sx

muls 2(a5),d4 ny\*sy

muls 4(a5),d5 nz\*sz

add.l d4,d3

add.l d5,d3 scalar product

bmi visible negative if surface visible

\* it is hidden

move.w #$20,(a6)+ set illumination for hidden

ill\_tidy

addq.w #6,a5 update normals pointer

move.w (a3)+,d5 current num edges

addq #1,d5 first vertex is repeated

add d5,d5 2 bytes per word

adda.w d5,a4 update edge list pointer

dbra d7,next\_ill

bra set\_colr

\* The surface is visible so find illumination level.

visible

move.w d0,d3 copy illum vector

move.w d1,d4

move.w d2,d5

muls (a5),d3 nx\*illx

muls 2(a5),d4 ny\*illy

muls 4(a5),d5 nz\*illz

add.l d4,d3

add.l d5,d3 -2^28<scalar prod <+2^28

add.l #$11100000,d3 0 < scalar prod < 2^29

move.w #24,d4

lsr.l d4,d3

cmp.w #$1f,d3 keep in range 0 to $1f

ble vis\_1 correct

move.w #$1f,d3 for

bra ill\_save errors

vis\_1

cmp.w #0,d3

bge ill\_save

clr d3

ill\_save

move.w d3,(a6)+ save it

bra ill\_tidy next ...

\*

\*

set\_colr

move.w npoly,d7

subq.w #1,d7

move.w illkey,d0 how many shades per colour

lea slumlst,a0 levels of illumination

lea srf\_col,a1

lea col\_lst,a2 colour for display

move.w #5,d6

sub.w d0,d6 5-illkey

next\_col

move.w (a0)+,d1 next illumination

cmp.w #$1f,d1 is it hidden

ble set\_col no

move.w #$20,(a2)+ it is, set flag

addq.l #2,a1 point to next intrinsic colour

bra set\_next

set\_col

lsr.w d0,d1 divide by 0, 2 or 4

move.w (a1)+,d2 the intrinsic colour

rol.b d6,d2 0 or 0, 16 or 0,8,16,24 = base

add.w d1,d2 illumination + colour base

bgt pass\_col

move.w #1,d2 avoid background

pass\_col

move.w d2,(a2)+ = final colour

set\_next

dbra d7,next\_col

rts

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

transfer

move.w my\_npoly,d7

move.w d7,npoly

subq.w #1,d7 counter

move.w d7,d0

lea my\_nedges,a0

lea snedges,a1

lea intr\_col,a2 intrinsic colours

lea srf\_col,a3 program intrinsic colours

loop0

move.w (a0)+,(a1)+ transfer edge numbers

move.w (a2)+,(a3)+ transfer intrinsic colours

dbra d0,loop0

\* calculate the number of vertices altogether

move.w d7,d0

lea my\_nedges,a6

clr d1

clr d2

loop1

add.w (a6),d1

add.w (a6)+,d2

addq #1,d2

dbra d0,loop1

\* move the edge list

subq #1,d2 counter

lea my\_edglst,a0

lea sedglst,a1

loop2

move.w (a0)+,(a1)+

dbra d2,loop2

\* and the coords list

move.w d1,oncoords

subq.w #1,d1

lea ocoordsx,a1

lea my\_datax,a0

lea ocoordsy,a3

lea my\_datay,a2

lea ocoordsz,a5

lea my\_dataz,a4

loop3

move.w (a0)+,(a1)+

move.w (a2)+,(a3)+

move.w (a4)+,(a5)+

dbra d1,loop3

\* and the window limits

move.w my\_xmin,clp\_xmin

move.w my\_xmax,clp\_xmax

move.w my\_ymin,clp\_ymin

move.w my\_ymax,clp\_ymax

rts

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* normalise a vector: unormalised components in d0,d1,d2

\* return normalised components

nrm\_vec

\* save the component squares

move d0,d3

move d1,d4

move d2,d5

muls d0,d0

muls d1,d1

muls d2,d2

\* sum of squares

add.l d1,d0

add.l d2,d0

\* calculate the magnitude

bsr sqrt

\* multiply the components by 2^14

move.w #14,d7

ext.l d3

ext.l d4

ext.l d5

lsl.l d7,d3

lsl.l d7,d4

lsl.l d7,d5

\* divide by magnitude to derive normalised components

divs d0,d3

divs d0,d4

divs d0,d5

\* return normalised components

move.w d3,d0

move.w d4,d1

move.w d5,d2

rts

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Find the sqrt of a long word N in d0 in three iterations: sqrt=1/2(squrt+N/squrt)

\* approximate starting value found from highest bit in d0: Result passed in d0.W

sqrt

tst.l d0

beq sqrt2 quit if zero

move.w #31,d7 31 bits to examine

sqrt1

btst d7,d0 is this bit set?

dbne d7,sqrt1

lsr.w #1,d7 bit is set: 2^d7/2 approx root

bset d7,d7 raise 2 to this power

move.l d0,d1

divs d7,d1 N/squrt

add d1,d7 squrt+N/squrt

lsr.w #1,d7 /2 gives new trial value

move.l d0,d1 N

divs d7,d1

add d1,d7

lsr.w #1,d7 second result

move.l d0,d1

divs d7,d1

add d1,d7

lsr.w #1,d7 final result

move.w d7,d0

sqrt2

rts

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* data\_04.s

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

include data\_03.s

ill\_vecx dc.w -100

ill\_vecy dc.w -16384 ;LIGHT SHINING FROM +Y TO -Y

ill\_vecz dc.w 0

vwpointz dc.w -100

illkey dc.w 2

intr\_col dc.w 0,1,0,0,1,1

OTHER\_PALETTE EQU 1 ;to use with illumination

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* bss\_04.s \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

include bss\_03.s

\* VARIABLES FOR SURFACE ILLUMINATION AND COLOUR

snormlst ds.w 100

slumlst ds.w 40

srf\_col ds.w 40

8. 3D'de Genel Dönüşümler

Bu bölümde 3 boyutlu yapıların manipülasyonunda yer alan çeşitli türdeki bir dizi dönüşümü inceleyeceğiz.

8.1. Geometrik Dönüşümler

Basit döndürme ve yer değiştirme kombinasyonları, farklı konumlarda ve farklı yönelimlerde bulunan birkaç grafik ilkelinden oluşan karmaşık bir sahnenin inşasında yaygın olarak kullanılır. Bu örnek dönüşümlerinin yanı sıra, kullanılabilecek daha egzotik bozulmalar da vardır. Yapılar çeşitli şekillerde manipüle edilebilir:

rotasyon - yönelim değişikliği,

kesme - bozulma,

ölçekleme - boyutta değişiklik,

yansıma - ayna görüntüsüyle yer değiştirme,

ters çevirme - içten dışa ve arkadan öne,

Genel olarak, herhangi bir 3x3 matris ölçekleme ve kesme kombinasyonunu üretecektir. Hacimde hiçbir değişiklik olmadığı özel durumda, sonuç saf bir dönüş olur. Bazen sabit (basit) matris elemanlarına sahip kesmeler, sabit açılarla dönüşü simüle etmek için kullanılır. Bu dönüşümlerin ilk üçü bu bölümde, klavyeden ve kumanda çubuğundan giriş ve kontrol ile gösterilmiştir.

Bu tür dönüşümler matrisler kullanılarak kolayca uygulanabilir ve bunların birçoğu, noktaları gerçekten dönüştürmeden önce bireysel matrislerin birleştirilmesi (çarpılması) yoluyla birleştirilebilir. Çok sayıda nokta söz konusu olduğunda, bu her dönüşümü ayrı ayrı gerçekleştirmeye kıyasla çok zaman kazandırır. Bunun bir örneği programlarda gösterilmiştir.

8.1.1. Rotasyonlar

Joystick hareket ettirildiğinde veya bir tuşa basıldığında ekranda karşılık gelen bir dönüş görmek isteriz. Prensip olarak, bunu yapmak çok basittir. Örneğin, joystick'in sola doğru hareketi x ekseni etrafında pozitif bir dönüşe neden olabilir ve sağa doğru hareketi negatif bir dönüşe neden olabilir. Diğer joystick hareketleri diğer eksenler etrafında dönüşler üretebilir. x, y ve z eksenleri etrafındaki basit dönüşler için matrisler Bölüm 6'da listelenmiştir.

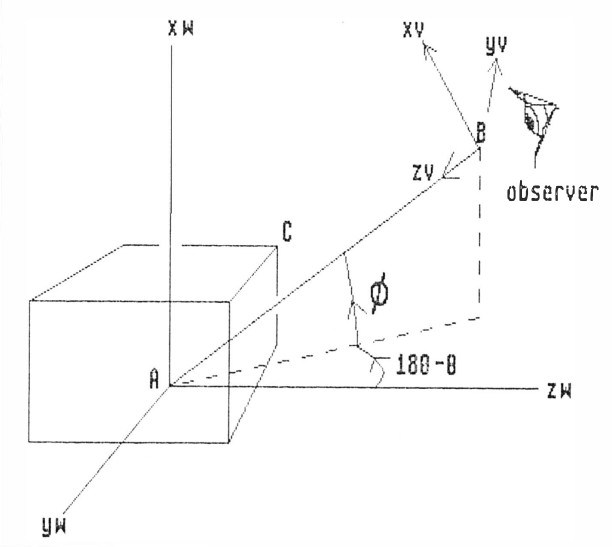
Joystick'in her hareketinden sonra, eski köşeleri uygun dönüş matrisiyle çarparak yeni bir nesne köşeleri kümesi oluşturulabilir. Bu şekilde, önceki dönüşün sonuçları bir sonrakinin başlangıç ​​noktası olarak kullanılabilir. Bunu yapmanın sorunu, dönüşümlerde ikili aritmetiğin yapıldığı doğruluktaki hataların kareden kareye birikmesi ve sonunda resmi kaosa sürüklemesidir. Bu soruna bir çözüm, nesneyi her seferinde bir referans konumundan (nesne karesi gibi) yeniden çizmektir; bu konum, joystick hareketlerine ayak uydurmak için nesneyle birlikte sürekli olarak döndürülen bir dizi "işaret levhasında" (yine birim vektörler) depolanan bilgilerle oluşturulur. Daha sonra nesne her seferinde yalnızca bir kez dönüştürülür. Bu yöntem, gözlemci serbestçe hareket ettiğinde görüntüleme dönüşümünde esastır. Bu konu bir sonraki bölümde ayrıntılı olarak ele alınacaktır.

Alternatif olarak, dönüşleri uygulamanın basit bir yolu vardır, ancak boylam ve enlem çizgilerini içeren şemaya benzer bir şema ile belirlenen bir hareketle, y ve x eksenleri etrafındaki dönüşler ayrı ayrı toplanır ve sonunda bir araya getirilir. Bu şemada, joystick'in (diyelim ki) birkaç hareketi hem sola hem de sağa (x ekseni etrafında dönüş) ve yukarı ve aşağı (y ekseni etrafında dönüş) herhangi bir sırayla gerçekleşmiş olabilir, ancak yalnızca ayrı toplamlar kaydedilir. Joystick'in tek bir hareketi, o yönde 1°'lik bir artışa karşılık gelebilir.

Örnek olarak, y ekseni etrafındaki toplam dönüşün 40° ve x ekseni etrafındaki toplam dönüşün 83° olduğunu varsayalım. O zaman toplam dönüş, y ekseni etrafında 40°'lik tek bir dönüş ve ardından x ekseni etrafında 83°'lik tek bir dönüş olarak alınır. Bunun, önce x ekseni etrafında ve sonra y ekseni etrafında dönmekle aynı şey olmadığını ve farklı bir sonuç verdiğini unutmayın. Sıranın önemli olması, dönüşlerin tuhaf bir özelliğidir. Dönüşlerin matris olarak yazılabilmesi, çarpma sırasının da matrislerin bir özelliği olduğu anlamına gelir.

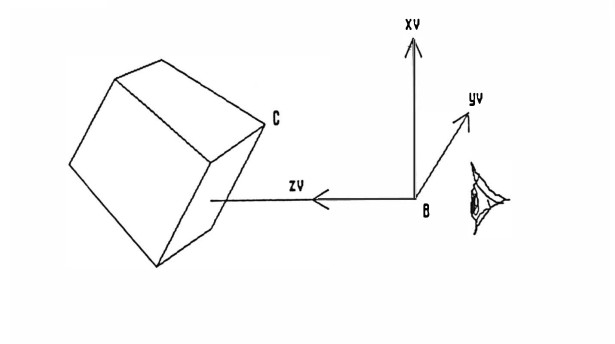
Önce y ekseni etrafında bir dönüş, ardından x ekseni etrafında bir dönüş yaparak, belgeler her seferinde aynı yönelime ulaşmak için bir tarif sunar. Bu, tıpkı boylam ve enlem dairelerini benzersiz bir şekilde kullanarak küre üzerinde bir konum bulmaya benzer. Y ekseni etrafındaki ilk dönüş enlem açısını verir ve x ekseni etrafındaki ikinci dönüş boylam açısını verir. Bu, bir nesneyi yönlendirmek için basit bir şema ile sonuçlanır, ancak göreceğimiz gibi, ekranda olan biten toplam mevcut dönüş açılarına bağlı olduğundan joystick tepkisi garip görünüyor.

Eğer bu kafa karıştırıcı görünüyorsa, nesneyi hareketsiz bırakıp gözlemciyi nesneden sabit bir uzaklıkta farklı yönelimlere hareket ettirmenin tamamlayıcı şemasını düşünün. Bu, bu bölümdeki örnek programda yapılan şeydir. Şekil 8.1, dünya çerçevesinde neler olup bittiğini göstermektedir. Nesne ile gözlemci arasında uzun bir direk, AB, hayal edebilirsiniz ve gözlemci direkten aşağıya nesneye doğru bakar. Gerçekleşen dönüşler direğin yönelimini değiştirir. Örnek programda, joystick'in sola veya sağa hareketi 9 değiştirir ve yukarı veya aşağı hareketi <J> değiştirir. Şimdi nesneyi döndürmenin tam tersiyle ilgileniyoruz.



*Şekil 8.1 Gözlemcinin nesne etrafında döndürülmesi*

Gözlemci şekilde gösterilen açılardadır ve ne gördüğünü bulmamız gerekir. Çizildiği gibi, gözlemci tepe noktası C'ye en yakındır ve onu oldukça iyi bir şekilde önden görür, bu nedenle gözlemcinin referans çerçevesinde (kutbun yatay olduğu yerde) her şey Şekil 8.2'deki gibi görünür. Bu görünüm yalnızca 0 ve <)> açılarını ve AB mesafesini bilerek nasıl oluşturulabilir? Dönmelerle ilgili çoğu problem gibi göründüğünden daha kolaydır ve geometrik (nesneyi hareket ettirme) ve koordinat (gözlemciyi hareket ettirme) dönüşümlerinin tamamlayıcı doğasıyla çok ilgisi vardır; bunlar Ek 6'da kapsamlı bir şekilde ele alınmıştır.



*Şekil 8.2 Gözlemcinin gördüğü görünüm*

Sorun, AB doğrusunun dünya eksenleri etrafındaki hangi dönüşlerinin onu zw ekseniyle aynı çizgiye getirdiğini bularak çözülür. Bunu yapmak için dönüş dizisi şudur:

1. xw etrafında (-0) döndürerek xw-zw düzlemine getirin,
2. (-()>) ile yw etrafında döndürerek zw ekseni boyunca getirin,

(3. xw'yi "yukarı" yön yapmak için zw etrafında (-y) döndürün).

Bu son adım, aslında programda uygulanmadığı için parantez içine alınmıştır, yani gözlemcinin herhangi bir "dönüşü" söz konusu değildir.

Eğer bu dönüş dizisi, izleyici dünya çerçevesi zw ekseni boyunca sabit bir konumdayken nesneye gerçekten uygulanırsa, genel sonuç aynı olur. Örnek programda yapılan tam olarak budur. Nesneye merkezi etrafında uygulanması gereken dönüş dizisi, sırayla (sağdakinin önce etki ettiğini unutmayın):

r rahat siny 0\ I COS(J) 0 -sin({>' 1 0 0 -siny rahat 0 0 1 0 0 cos0 sin0 veya 0 1 , sin4> 0 COS()> , veya -sin0 COS0 ,

çarpıldığında (birleştirildiğinde) programda görünen elemanların tek matrisini verir. Tüm köşeleri bu matrisle dönüştürdükten sonra geriye kalan tek şey her z koordinatına AB (ayrıca Ovz olarak da adlandırılır) mesafesini eklemektir.

Bu özellikle basit dönüşümü gözlemcinin referans çerçevesine Bölüm 10'da, açıların (Euler açıları olarak adlandırılır) joystick hareketiyle kolayca ilişkilendirilebildiği bir uçuş simülatöründe tekrar kullanacağız. Açıların tanımlanma şeklinin kısıtlanmasını sorun etmiyorsanız sorun değil. Genel olarak, daha fazla özgürlük istenebilir.

8.1.2. Ölçeklendirme

Ölçekleme çok basittir. Nesneyi basitçe daha büyük veya daha küçük yapar. Ölçek değişikliği üç eksen boyunca bağımsız olarak gerçekleşir. Üç eksen boyunca farklı ölçek faktörleri, a, b ve c ile genel bir ölçek değişikliği için dönüşüm matrisi şudur:

a 0 0 ' 0 b 0 0 0 c 1

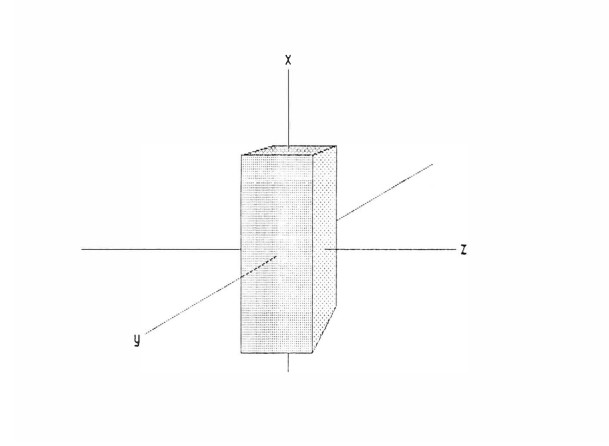
Hem b hem de c birlikse ve a birlikten büyükse, ortaya çıkan bozulma x ekseni boyunca bir gerilmedir. Örnek programda uygulanan budur. Şekil 8.3'te gösterilmiştir.

8.1.3. Kesme

Bir kesme bozulması, bir yüzün karşısına göre yer değiştirmesi etkisine sahiptir. En basit durumda, koordinatlardan biri diğerlerinden birine orantılı olarak artırılır. Eğer x, z'ye orantılı olarak artarsa, matris şu şekildedir:

1 0 1 0 1 0 , 0 0 1

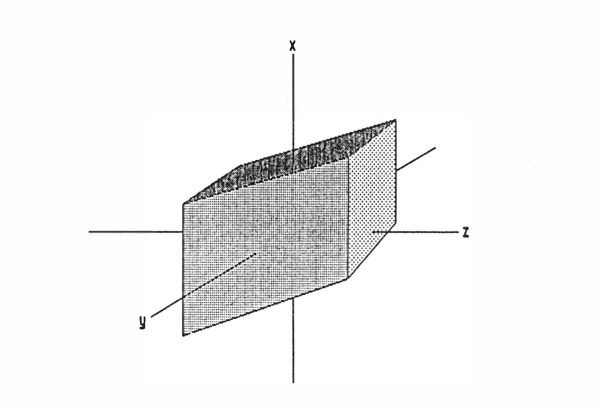
ve hem y hem de z değişmeden kalır. Bu Şekil 8.4'te gösterilmiştir ve örnek programa dahil edilmiştir. x



*Şekil 8.3 X ekseni boyunca bir gerilme*

x hem y hem de z'ye orantılı olarak artarsa ​​bozulma daha egzotik hale gelir. Bu Şekil 8.5'te gösterilmiştir ve ayrıca örnek programa dahil edilmiştir. Matris

1 1 1 0 1 0 0 0 1



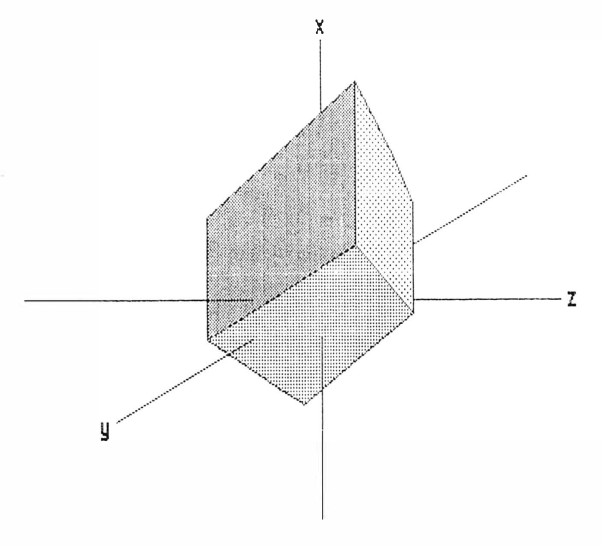
*Şekil 8.4 Z'ye orantılı x yönünde bir kesme*

8.2. Örnek Dönüşümleri

Şimdiye kadar hareket 3 boyutlu olmasına rağmen, gösterilen tek yapı düz A monolitiydi. Şimdi, bu tür altı monolit bir araya getirilerek bir A küpü oluşturuluyor.

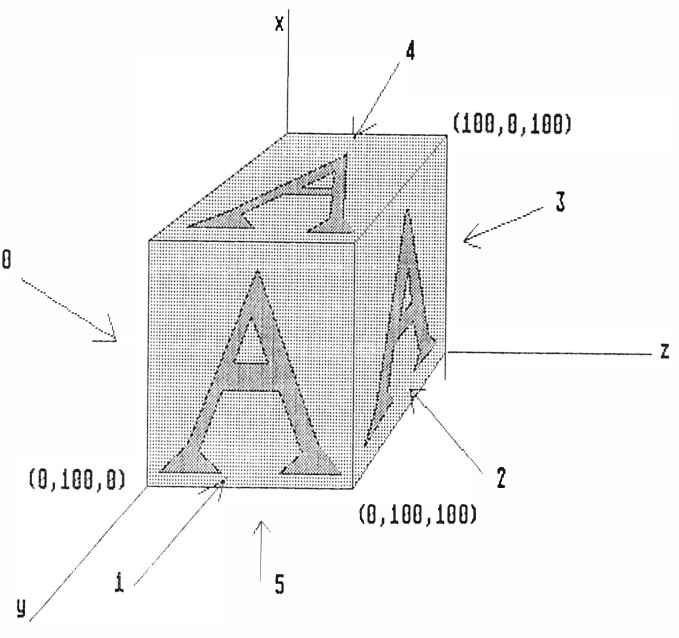
Örnek dönüşümleri genellikle dünya uzayında ilkelleri ayarlayan yönelim ve konum değişiklikleri anlamına gelir ve bu terimi A küpünü oluşturan işlemler kümesini tanımlamak için kullanırız. Oluşturulduktan sonra küp, tartıştığımız dönüşümleri göstermek için bir temel olarak kullanılabilir.

Bu şekilde bir küp inşa etmek için önce yw-zw düzleminde bir monolit yatırılır ve sonra diğer kenarları oluşturmak için beş kez daha döndürülür ve yer değiştirilir.



*Şekil 8.5 Hem y hem de z'ye orantılı x yönündeki bir kesme*

Bu, kenarların numaralandırıldığı Şekil 8.6'da gösterilmiştir. Altı kenarın dönüş açıları ve yer değiştirmeleri, veri dosyası data 05.s'deki instjmgles ve inst\_disp listelerindedir ve 0, 0, 8 ve x, y, z sırasıyladır.



*Şekil 8.6: Bir A küpünün inşası*

8.3. Fiziksel Gerçekçilik

Fiziksel nesnelerin şekil ve renkten daha incelikli nitelikleri vardır. Bu, özellikle hareket meydana geldiğinde belirgindir. Gerçek nesneler bir yerden bir yere anında hareket etmezler veya hareket başladığı anda son hızlarına ulaşmazlar. Hız maksimum değerine ulaşırken bir ivmelenme periyodu vardır. Benzer şekilde gerçek bir nesne hızını anında sıfıra düşüremez. Bir yavaşlama periyodu gereklidir. Hızlanma ve yavaşlama, her ikisi de fiziksel bir nesnenin ek bir niteliğinin, eylemsizliğinin veya kütlesinin kanıtıdır. Bir nesnenin kütlesi, ne kadar hızlı hızlandırılabileceğini veya durdurulabileceğini belirler. Fiziksel nesnelerin gerçekçi bilgisayar modellerini oluştururken bu ayrıntılara dikkat etmek önemlidir. Bir cismin kütlesinin hareketini belirlemedeki rolü, Newton'un Hareket Yasaları'nda gerçekten özetlenmiştir. Özünde, bir cisim bir kuvvet tarafından etki altına alındığında kuvvetle orantılı olarak ivmeleneceğini ve kuvvet yoksa sabit hızda (veya hareketsiz) kalacağını söylerler.

Örnek programlarda, joystick hareketlerini uygulanan kuvvetler olarak modelleyerek bu yasaları dahil etmeye yönelik bazı girişimlerde bulunulmuştur. Sonuç olarak, görüntünün hareketi hemen değil, eylemsizliği tarafından belirlenen bir ivmeyle gerçekleşir. Ayrıca, uygulanan kuvvet kaldırıldığında hızın sıfıra düşmesi ve sürekli uygulandığında bile hızın maksimum olması için sürtünme etkisi dahil edilmiştir. Programlarda, hareket tamamen dönmeli olsa da aynı ilkeler geçerlidir.

8.4. Klavye ve Joystick'ten Giriş

Gözlemcinin programla düzgün bir şekilde etkileşime girebilmesi için programın akışını değiştirebilmesi gerekir. Aksi takdirde, program ne kadar karmaşık olursa olsun, tamamen deterministiktir. Bunun anlamı, bir başlangıç ​​koşulundan başlanarak, programın sonunun tamamen belirlenmiş olmasıdır. Bilgisayardaki sözde "rastgele sayı üreteci" bile, çok sayıda olası sonucu olmasına rağmen deterministiktir ve rastgele görünür. Gerçek dünya rastgele gibi görünür ancak geçmişi yeniden çalıştıramayacağınız için kimse kesin olarak bilemez!

Programla etkileşime girmenin en basit yolları klavye ve fareden veri girmektir, ancak yakında hepimiz stereoskopik görüntüleyiciler ve dokunsal sensörlerden oluşan kulaklıklar takıyor olacağız. Sanal Gerçeklik çağı geldi. Birisi doğrudan beyne nasıl bağlanacağını bulduğunda işler gerçekten ilginçleşecek. Şu anda, göreve ayrılmış bellekte belirli kayıtlar olduğu için klavyeden ve kumanda kolundan girdi almaya razı olacağız, çünkü bu basittir.

8.4.1. Klavye

Klavyeyi doğrudan kütüphane fonksiyonları aracılığıyla okumanın birkaç yolu vardır. Bunlardan hiçbirini kullanmayacağız. Sistem Kayıtlarına doğrudan erişimimiz olduğundan, bilgi için doğrudan oraya gideceğiz.

F1Özellikle fonksiyon tuşlarını okumak istiyoruz F7. Bilgi SbfecOl adresindeki bayttadır. Bu tuşlardan birine basıldığında bayt aşağıdaki gibi olacaktır:

ANAHTAR f1 f2 13 f4 15 16 17

KUYRUK 51 $5d $5b $59 $57 $55 $53

Yapmamız gereken tek şey onu okumak ve ona göre hareket etmektir.

8.4.2. Joystick

Joystick ayrıca doğrudan bellekteki bir adresten de okunabilir; bu durumda SdffOOc'deki kelimedir. Joystick'i okumak klavyeden biraz daha karmaşıktır ancak bunun tek nedeni dört olasılık olmasıdır: sol, yukarı, sağ veya aşağı.

Bu şu şekilde çalışır:

BİTLER AYARI sadece 8 sadece 8 ve 9 sadece 0 sadece 0 ve 1

YÖN yukarı sol aşağı sağ

Uçların incelenmesi, joystick'in hangi yöne hareket ettirildiğinin kolayca test edilmesini sağlar.

8.5. Örnek Program

Program, joystick kontrolü altında her yüzünde A harfi yazılı bir küpü dönüşümlü olarak gösterir. Ayrıca küp, aşağıda ayrıntılı olarak açıklandığı gibi fonksiyon tuşlarına basılarak dönüş gerçekleşirken kesme ve ölçekleme dönüşümlerine tabi tutulabilir.

8.5.1. transfms.s

Bu kontrol programıdır. Değişkenleri başlattıktan sonra, dönüş hızını, görüntüleme mesafesini ve bir kayma veya ölçek değişikliğinin gerçekleşip gerçekleşmeyeceğini seçmek için joystick ve klavye ayarlarını okur. Bu son dönüşümlerin her ikisi de kelime boyutu değişkenlerini aralık içinde tutmak için bir boyut küçültmeyle birlikte gelir.

Giriş tamamlandığında küp, tüm kenarlar için çoklu nesne-dünya dönüşümüyle dünya çerçevesinde döndürülmemiş veya bozulmuş bir şekilde bir araya getirilir. Bunu takiben bozulma, perspektif projeksiyonu için köşeleri dönüştüren genel dönüşümü üretmek üzere görüntüleme dönüşümüyle birleştirilir.

8.5.2. çekirdek\_05.s

İşte yeni alt rutinler. İlk kısım, v9, vp ve vy görüntüleme açılarından dönüş dönüşümünü oluşturmak ve sonra (kesme ile birleştirildikten sonra) köşeleri dönüştürmek için kullanmakla ilgilidir. Bunu takiben rutinler, joystick ve klavyeyi okumak ve buna göre ayarlamalar yapmakla ilgilidir.

Eylemsizliği simüle etmek için, joystick'in hareketleri dönüş açılarına değil, dönüş açılarına, maksimuma kadar, artışlar olarak dönüştürülür. Bu artışlar, toplam dönüş açılarını vermek için her seferinde açılara eklenir.

Ek olarak, artışlar her seferinde 1 azaltılarak yerleşik sürtünmeli yavaşlama sağlanır. Joystick alternatiflerini uygulama prosedürü, çeşitli olası alt rutinlere bir vektör atlama tablosu kullanır. Bu, uzun bir listedeki her olasılığı test etmekten kaçınmanın zarif bir yoludur. Bu teknik ayrıca klavye girişi için de kullanılır.

Tamamen fonksiyon tuşlarıyla ilgili olan yedi olası klavye girişi vardır F1:F7

F1(sürekli olarak) asgari bir mesafeye yaklaşmak,

F2uzaklaşmak (sürekli olarak),

F3kesme 1'i uygula (x, z ile artar, xkesme olarak adlandırılır),

F4kesme 2'yi uygula (x, y ve z ile artar, ykesme olarak adlandırılır),

F5bir germe uygulayın (y ve z 1/2 oranında azaltılır),

F6hareketi durdurmak ( F1ve F2)

F7quit - sistemi sıfırla.

F3, F4ve girdisi F5, bit-change talimatını kullanarak bir kelime uzunluğu bayrağının, shearflg, alt üç bitini geçişli bir şekilde ayarlamak için kullanılır. Bu, dönüşümün uygulanıp uygulanmaması gerektiğine dair bir kayıt sağlamak için uygun bit DEĞİLDİR. Hangi bayrak bitlerinin arc set edildiğini inceleyen rutin, aslında hiçbir şey için kullanılmayan ve diğer dönüşümleri denemek için kullanılabilen kombinasyonların seçeneğini de içerir (ürünler, birleştirme ürünlerinde kelime boyutunu aşmadığı sürece).

Son olarak kesme ve dönme matrisleri, küp üzerinde etki edecek genel dönüşümü üretmek için çarpılır.

8.5.3. bss\_05.s

Bu bölüm için yeni değişkenler.

8.5.4. data\_05.s

Bu bölüm için yeni veriler. Özellikle, kesme ve gerilme için 3x3 matrislerin, matris birleştirme rutinini basitleştirmek için sütun sırasına göre düzenlendiğine dikkat edin.

\*

\* trnsfrrms.s

\* Various transforms

\*SECTION TEXT

opt d+

bra main

include systm\_00.s

include core\_05.s motion of the view frame

main bsr init

\* transer all the data

bsr transfer

move.w oncoords,vncoords

move.w vncoords,wncoords

\* Initialise dynamical variables

move.w #-50,Ovx view frame initial position

move.w #0,Ovy

move.w #150,Ovz intialise rotation angles to zero

clr.w vphi

clr.w vgamma

clr.w shearflg set flag to no shear

move.w #25,vtheta\_inc initial rotation rates

move.w #25,vphi\_inc

clr.w speed

clr.w screenflag 0=screen 1 draw, 1=screen 2 draw

loop4:

\* Switch the screens each time round

tst.w screenflag screen 1 or screen 2?

beq screen\_1 draw on screen 1, display screen2

bsr drw2\_shw1 draw on screen 2, display screen1

clr.w screenflag and set the flag for next time

bra screen\_2

screen\_1:

bsr drw1\_shw2 draw on 1, display 2

move.w #1,screenflag and set the flag for next time

screen\_2:

\* look for changes in the rotation angles

bsr joy\_in

\* check function keys for a shear or a change the speed

bsr key\_in

\* Adjust to new rotation angles and speed

bsr angle\_update

bsr speed\_adj

\* Construct compound object from same face at different position

move.w nparts,d7 how many parts in the object

subq #1,d7

lea inst\_angles,a0 list of angles for each part

lea inst\_disp,a1 ditto displacements

\* Do one face at a time

instance:

move.w d7,-(sp) save the count

move.w (a0)+,otheta next otheta

move.w (a0)+,ophi next ophi

move.w (a0)+,ogamma next ogamma

move.w (a1)+,Oox next displacements

move.w (a1)+,Ooy

move.w (a1)+,Ooz

movem.l a0/a1,-(sp) save position in list

bsr otranw object to world transform

bsr wtranv\_1 construct the rotation transform

bsr shear concatenate with shear (if flag set)

bsr wtranv\_2 and transform the points

bsr illuminate if it is visible find the shade

bsr perspective

bsr polydraw draw the face

movem.l (sp)+,a0/a1 restore pointers

move.w (sp)+,d7 for all the parts of the object

bra loop4

\*SECTION DATA

include data\_00.s

include data\_03.s

include data\_05.s

\*SECTION BSS

include bss\_05.s

END

tst.w screenflag screen 1 or screen 2?

beq screen\_1 draw on screen 1, display screen2

bsr drw2\_shw1 draw on screen 2, display screen1

clr.w screenflag and set the flag for next time

bra screen\_2

screen\_1:

bsr drw1\_shw2 draw on 1, display 2

move.w #1,screenflag and set the flag for next time

screen\_2.

:

\* look for changes in the rotation angles

bsr joy\_in

\* check function keys for a shear or a change the speed

bsr key\_in

\* Adjust to new rotation angles and speed

bsr angle\_update

bsr speed\_adj

\* Construct compound object from same face at different position

move.w nparts,d7 how many parts in the object

subq #1,d7

lea inst\_angles,a0 list of angles for each part

lea inst\_disp,a1 ditto displacements

\* Do one face at a time

instance:

move.w d7,-(sp) save the count

move.w (a0)+,otheta next otheta

move.w (a0)+,ophi next ophi

move.w (a0)+,ogamma next ogamma

move.w (a1)+,Oox next displacements

move.w (a1)+,Ooy

move.w (a1)+,Ooz

movem.l a0/a1,-(sp) save position in list

bsr otranw object to world transform

bsr wtranv\_1 construct the rotation transform

bsr shear concatenate with shear (if flag set)

bsr wtranv\_2 and transform the points

bsr illuminate if it is visible find the shade

bsr perspective

bsr polydraw draw the face

movem.l (sp)+,a0/a1 restore pointers

move.w (sp)+,d7 for all the parts of the object

bra loop4

\*SECTION DATA

include data\_00.s

include data\_03.s

include data\_05.s

\*SECTION BSS

include bss\_05.s

END

\_

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Core\_05.s

\* A set of subroutines for transforming world coords. Including rotations of vtheta

\* vphi and vgamma about the x,y and z axes as well as x, y and z shears.

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

include Core\_04.s

\* The matrix for the rotations is constructed.

\* convert rotation angles to sin & cos and store for rotation matrix.

wtranv\_1

bsr view\_trig find the sines and cosines

\* construct transform matrix wtranv.

lea stheta,a0

lea ctheta,a1

lea sphi,a2

lea cphi,a3

lea sgamma,a4

lea cgamma,a5

lea w\_vmatx,a6

\* do element WM11

move.w (a3),d0 cphi

muls (a5),d0 cphi\*cgamma

lsl.l #2,d0

swap d0

move.w d0,(a6)+ WM11

\* do element WM12

move.w (a1),d0 ctheta

muls (a4),d0 ctheta\*sgamma

move.w (a0),d1 stheta

muls (a2),d1 stheta\*sphi

lsl.l #2,d1

swap d1

muls (a5),d1 stheta\*sphi\*cgamma

add.l d0,d1 stheta\*sphi\*cgamma + ctheta\*sgamma

lsl.l #2,d1

swap d1

move.w d1,(a6)+

\* do WM13

move.w (a0),d0 stheta

muls (a4),d0 stheta \* sgamma

move.w (a1),d1 ctheta

muls (a2),d1 ctheta\*sphi

lsl.l #2,d1

swap d1

muls (a5),d1 ctheta\*sphi\*cgamma

sub.l d1,d0 stheta\*sgamma - ctheta\*sphi\*cgamma

lsl.l #2,d0

swap d0

move.w d0,(a6)+

\* do WM21

move.w (a3),d0 cphi

muls (a4),d0 ctheta\*sgamma

lsl.l #2,d0

swap d0

neg d0

move.w d0,(a6)+

\* do WM22

move.w (a1),d0 ctheta

muls (a5),d0 ctheta\*cgamma

move.w (a0),d1 stheta

muls (a2),d1 stheta\*sphi

lsl.l #2,d1

swap d1

muls (a4),d1 stheta\*\*sphi\*sgamma

sub.l d1,d0 ctheta\*cgamma-stheta\*sgamma

lsl.l #2,d0

swap d0

move.w d0,(a6)+

\* do WM23

move.w (a0),d0 stheta

muls (a5),d0 stheta\*cgamma

move.w (a1),d1 ctheta

muls (a2),d1 ctheta\*sphi

lsl.l #2,d1

swap d1

muls (a4),d1 ctheta\*sphi\*sgamma

add.l d0,d1

lsl.l #2,d1

swap d1

move.w d1,(a6)+

\* do WM31

move.w (a2),(a6)+

\* do WM32

move.w (a3),d0 cphi

muls (a0),d0 cphi\*stheta

lsl.l #2,d0

swap d0

neg d0

move.w d0,(a6)+

\* do WM33

move.w (a1),d0 ctheta

muls (a3),d0 ctheta\*cphi

lsl.l #2,d0

swap d0

move.w d0,(a6)+

rts

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* PART 2: Transform the World coords to view coords.

wtranv\_2

move.w wncoords,d7

ext.l d7 any to do?

beq wtranv3

subq.w #1,d7

lea wcoordsx,a0

lea wcoordsy,a1

lea wcoordsz,a2

lea vcoordsx,a3

lea vcoordsy,a4

lea vcoordsz,a5

exg a3,d3 save cos we're short of registers

link a6,#-6 save 3 words

wtranv1

moveq.l #2,d6 3 rows in matrix

lea w\_vmatx,a3 init max pointer

\* calculate the next wx, wy and wz

wtranv2

move.w (a0),d0 wx

move.w (a1),d1 wy

move.w (a2),d2 wz

sub.w #50,d0 wx-50

sub.w #50,d1 wy-50

sub.w #50,d2 wz-50

muls (a3)+,d0 wx\*Mi1

muls (a3)+,d1 wy\*Mi2

muls (a3)+,d2 wz\*Mi3

add.l d1,d0

add.l d2,d0 wx\*Mi+wy\*Mi2+wz\*Mi3

lsl.l #2,d0

swap d0

move.w d0,-(a6)

dbra d6,wtranv2 repeat for 3 elements

move.w (a6)+,d0

add.w Ovz,d0

move.w d0,(a5)+ becomes vz

move.w (a6)+,(a4)+

exg a3,d3 restore vx, save matx pointer

move.w (a6)+,d0

add.w #100,d0

move.w d0,(a3)+ becomes vx

exg a3,d3 save vx, restore matx pointer

addq.l #2,a0 point to next wx

addq.l #2,a1 wy

addq.l #2,a2 wz

dbra d7,wtranv1 repeat for all ocoords

unlk a6 close frame

wtranv3

rts

\*

\* Calculate the sines and cosines of the view angles

view\_trig

move.w vtheta,d1 theta

bsr sincos

move.w d2,stheta sine

move.w d3,ctheta cosine

move.w vphi,d1

bsr sincos

move.w d2,sphi

move.w d3,cphi

move.w vgamma,d1 gamma

bsr sincos

move.w d2,sgamma

move.w d3,cgamma

rts

\*

\* Read jstick and update vars accordingly.

joy\_in

move.w $dff00c,d0 read jstick register

\* convert value to angle totals

angle\_speed

btst #8,d0 up or left?

beq dwn\_rt nope

btst #9,d0 left?

beq up

bra left

dwn\_rt

btst #0,d0 down or right?

beq joy\_out

btst #1,d0 right?

beq down

bra right

joy\_out

rts

IFD JOY1

\* set up the increments to angles +/-10 is the limit

up

subq.w #2,vphi\_inc

rts

down

addq.w #2,vphi\_inc

rts

left

addq.w #2,vtheta\_inc

rts

right

subq.w #2,vtheta\_inc

rts

ENDC

IFD JOY2

up

move.w #350,vyangle

bsr rot\_vy

rts

down

move.w #10,vyangle

bsr rot\_vy

rts

left

move.w #10,vxangle

bsr rot\_vx

rts

right

move.w #350,vxangle

bsr rot\_vx

rts

ENDC

IFD JOY3

up

bsr rot\_down

rts

down

bsr rot\_up

rts

left

bsr rot\_left

rts

right

bsr rot\_right

rts

ENDC

IFD JOY4

up

move.w #-5,vphi\_inc

rts

down

move.w #5,vphi\_inc

rts

left

move.w #5,vtheta\_inc

rts

right

move.w #-5,vtheta\_inc

rts

ENDC

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

angle\_update

move.w vtheta\_inc,d0

bmi vth\_neg

beq chk\_phi

subq.w #1,vtheta\_inc

cmp.w #25,vtheta\_inc

ble chk\_phi

move.w #25,vtheta\_inc

bra chk\_phi

vth\_neg

addq.w #1,vtheta\_inc

cmp.w #-25,vtheta\_inc

bge chk\_phi

move.w #-25,vtheta\_inc

chk\_phi

move.w vphi\_inc,d0

bmi vph\_neg

beq chk\_out

subq.w #1,vphi\_inc

cmp.w #25,vphi\_inc

ble chk\_out

move.w #25,vphi\_inc

bra chk\_out

vph\_neg

addq.w #1,vphi\_inc

cmp.w #-25,vphi\_inc

bge chk\_out

move.w #-25,vphi\_inc

chk\_out

\* update vtheta

move.w vtheta,d0 the previous angle

add.w vtheta\_inc,d0 increase by increment

bgt thta\_1 check it lies between 0 and 360

add #360,d0

bra thta\_2

thta\_1

cmp.w #360,d0

blt thta\_2

sub #360,d0

thta\_2

move.w d0,vtheta becomes the current angle

\* update vphi

move.w vphi,d0

add.w vphi\_inc,d0

bgt phi\_1

add #360,d0

bra phi\_2

phi\_1

cmp.w #360,d0

blt phi\_2

sub #360,d0

phi\_2

move.w d0,vphi

rts

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

key\_in

in\_key

clr.w d0

move.b $bfec01,d0

cmp.b #$5f,d0

beq f1

cmp.b #$5d,d0

beq f2

cmp.b #$5b,d0

beq f3

cmp.b #$59,d0

beq f4

cmp.b #$57,d0

beq f5

cmp.b #$55,d0

beq f6

cmp.b #$53,d0

beq f7

rts

IFD JOY3

f1 bsr roll\_left

rts

f2 bsr roll\_right

rts

f3 move.w #-2,speed

rts

f4 move.w #2,speed

rts

f5 move.w #3,speed

rts

f6 move.w #0,speed stop

rts

f7 move.w #QUIT,quitflag

rts

ELSEIF

f1 move.w #-1,speed reverse

rts

f2 move.w #1,speed forward

rts

f3 bchg.b #2,shearflag toggle x shearflag

rts

f4 bchg.b #1,shearflag toggle yshearflag

rts

f5 bchg.b #0,shearflag toggle z shearflag

rts

f6 move.w #0,speed stop

rts

f7 move.w #QUIT,quitflag

rts

ENDC

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* concatenate the shear with the rotation

shear

clr d0

move.b shearflag,d0 flag is lower 3 bits

and #$f,d0

\* there are 8 possibilities 111 - 000, xyz respectively

lea shear\_jump,a0

lsl.w #2,d0 get offset

move.l 0(a0,d0.w),a0

jmp (a0)

shear\_jump

dc.l null,z,y,user1,x,user2,user3,user4

null

rts

z

lea zshear,a0

lea w\_vmatx,a1

bsr concat

rts

y

lea yshear,a0

lea w\_vmatx,a1

bsr concat

rts

user1

rts

x

lea xshear,a0

lea w\_vmatx,a1

bsr concat

rts

user2 rts

user3 rts

user4 rts

\*

\* Multiply two 3x3 matrices pointed to by a0 and a1

\* order is (a1)x(a0) with result sent to temp store at (a2)

\* (a0) is in column order whilw (a1) and (a2) are in row order, of word length elements.

\* Finally (a2) is copied to (A1).

concat

lea tempmatx,a2

move.w #2,d7 3 rows

conc1

move.w #2,d6

movea.l a0,a3 reset shear pointer

conc2

move.w (a1),d1

ext.l d1

lsr.l #1,d1

move.w 2(a1),d2

ext.l d2

lsr.l #1,d2

move.w 4(a1),d3

ext.l d3

lsr.l #1,d3

muls (a3)+,d1

muls (a3)+,d2

muls (a3)+,d3

add.w d2,d1

add.w d3,d1

move.w d1,(a2)+ next product element

dbra d6,conc2 do all elements in row

addq.w #6,a1 point to next row

dbra d7,conc1 for al rowa

\* transfer result back to rotation matrix

lea tempmatx,a0

lea w\_vmatx,a1

move.w #8,d7 num elements -1

conloop

move.w (a0)+,(a1)+

dbra d7,conloop

rts

\* set the velocity components

speed\_adj

move.w speed,d0

lsl.w #3,d0 scale it

move.w Ovz,d1

cmp.w #10,Ovz

bgt adj\_out

move.w #10,Ovz

adj\_out

add.w d0,Ovz

rts

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* bss\_05.s

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

include bss\_04.s

\* World Frame Variables

wncoords ds.w 1 num vertices in world frame

\* View frame vars

vtheta ds.w 1 rotation of view frame abouut wx

vphi ds.w 1 wy

vgamma ds.w 1 wz

Ovx ds.w 1 view frame x origin in world frame

Ovy ds.w 1

Ovz ds.w 1

\* General transform matrices

w\_vmatx ds.w 9

tempmatx ds.w 9

\* joystick

joy\_data ds.w 1

\* Dynamic vars

speed ds.w 1

vtheta\_inc ds.w 1

vphi\_inc ds.w 1

vgamma\_inc ds.w 1

shearflag ds.w 1

quitflag ds.w 1

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Data\_05.s

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TRANSFORM EQU 1

include data\_04.s

my\_datax dc.w 100,100,0,0,20,90,20,15,45,45,15,55

dc.w 70,55,10,10,10,10,20,20

my\_datay dc.w 0,100,100,0,15,60,87,25,40,65,74,46

dc.w 55,61,30,5,95,60,25,74

my\_dataz dc.w 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

xshear dc.w 1,0,0,0,1,0,1,0,1

yshear dc.w 1,0,0,1,1,0,1,0,1

zshear dc.w 2,0,0,0,1,0,0,0,1

nparts dc.w 6

inst\_angles dc.w 0,0,0,90,0,0,180,0,0,270,0,0,0,270,0,0,90,0

inst\_disp dc.w 0,0,0,0,100,0,0,100,100,0,0,100,100,0,0,0,0,100

9. Dünya Çapında Uçmak

9.1. Giriş

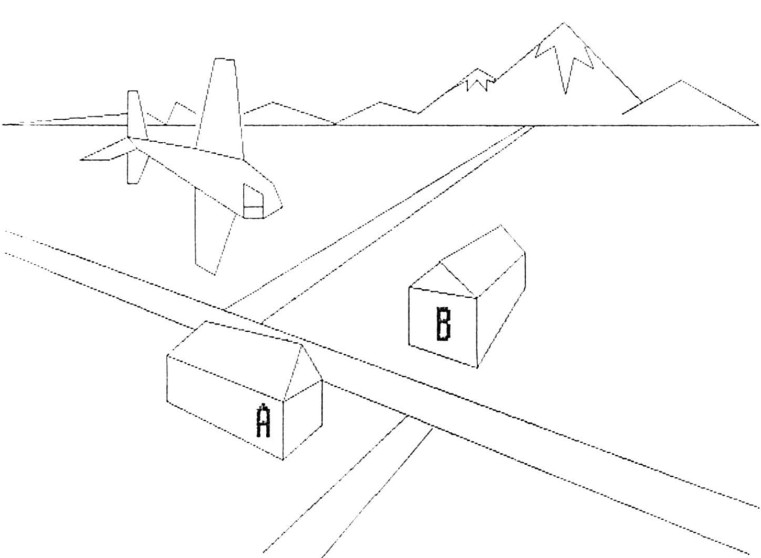
Bir uçuş simülatörü mü? Aslında tam olarak öyle değil ama oraya doğru gidiyoruz.

Gözlemcinin bağımsız hareketinin simülasyonunu tam olarak uygulamak için biraz daha vektör cebirine ihtiyacımız var. Görev, herhangi bir yönde hareket edebilen bir gözlemcinin bakış açısından dünya modelinin bir görünümünü oluşturmaktır. Bu, önceki bölümde kullandığımız basit prosedürden farklıdır. Şimdi bir kumanda kolunu çalıştırmak ve dünya çerçevesinde oluşturulan nesnelerin montajında ​​yolumuzu bulmak istiyoruz. Kumanda kolu ileri veya geri itildiğinde ekrandaki görünümün yukarı veya aşağı hareket etmesini ve kumanda kolu sağa veya sola hareket ettirildiğinde sola veya sağa hareket etmesini istiyoruz. Başka bir deyişle, ekrandaki tüm hareketler gözlemcinin mevcut konumuna göre olmalıdır. Bir uçağın pilotu baş aşağı uçuyor olsa bile, onun "yukarı" algısı kokpitin çatısına doğru yönlendirilir ve yerdeki biri için bu "aşağı"dır. Önemli olan, "yukarı", "aşağı", "sol" ve "sağ"a karşılık gelen tüm hareketlerin, görüş çerçevesi adını verdiğimiz gözlemcinin referans çerçevesine uygulanmasıdır. Önceki bölümde kullandığımız Euler açılarına göre dönüşün aksine, burada dönüşlerin görüntü çerçevesi eksenleri etrafında olmasını istiyoruz.

Daha spesifik olmak gerekirse, joystick geri çekildiğinde ne olmasını beklediğimizi soralım. Resmin dikey olarak yukarı doğru hareket etmesini bekliyoruz ve bu, gözlemcinin yönelimi ne olursa olsun her zaman gerçekleşmelidir. Uçağın veya kontrol edilen şeyin yatay olarak uçtuğu ancak kanatlarının dikey olduğu bir konuma geldiğimizi varsayalım.

Şekil 9.1 bu yönelimi göstermektedir.

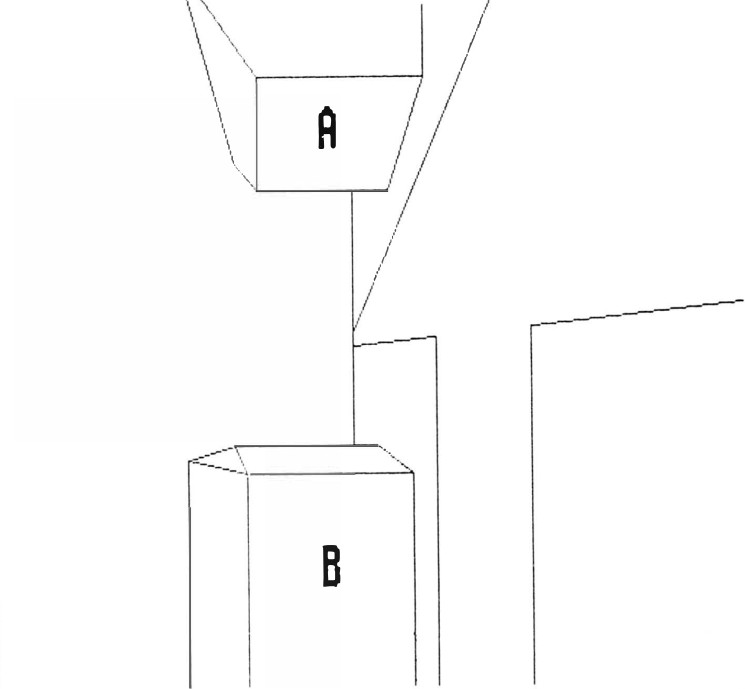
Joystick geri çekilirse, nesne A ekranın üst kısmında görüntüye girecek ve nesne B ekranın alt kısmında görüntüden çıkacaktır. Uçağın pilotunun gördüğü görüntü Şekil 9.2'de gösterilmiştir. Sorun burada yatmaktadır. Pilotun herhangi bir anda neyin "yukarı" neyin "aşağı" olduğuna dair çok belirgin bir algısı vardır ve bu kokpitte değişmese de, dışarıdaki dünyaya göre sürekli değişmektedir. Önceki bölümde "yukarı"yı v(J) artışına ve solu v9 artışına bağlamak kolaydı, ancak görüş çerçevesinden referans alındığında tüm bu hareketler gözlemcinin herhangi bir anda yönelimine bağlıdır.



*Şekil 9.1 Gözlemcinin (uçak) dünya görüşü*

Bu problemi çözmenin birden fazla yolu vardır. Bir yöntem, koordinatların görünüm çerçevesine dönüştürüldükten sonra dönüşlerini gerçekleştirmek için kontrol matrislerini kullanmaktır. Kontrol matrisleri görünüm x, y ve z eksenleri etrafında basit dönüşler gerçekleştirir. Bu yöntem bir sonraki bölümde kullanılır. Başka bir yol, dünya çerçevesindeki görünüm çerçevesinin konumunun ve yöneliminin sürekli bir kaydını tutmak ve joystick'in hareketlerinden kaynaklanan görünüm çerçevesinin hareketlerini üretmektir. Bu ikinci yöntem, gözlemcinin yolunu takip eden dönüşler ve ötelemeler geçiren bir dizi görünüm çerçevesi ekseni kavramına büyük ölçüde dayanır. Ayrıca, bu bölümde tanıtmak istediğimiz, dünya çerçevesindeki herhangi bir eksen etrafında dönüşler gerçekleştirmek için çok yararlı olan keyfi bir eksen etrafında dönüş kavramını da bünyesinde barındırır.

Elbette, Euler açılarının sınırlamalarını kabul ederek, daha basit yörünge benzeri bir şekilde görüntü çerçevesi yönelimini düzeltmeye karar verebiliriz. 10. Bölümde, bir uçuş simülatörünün bu yaklaşımların her birini kullanarak nasıl iyi çalıştığını gösteriyoruz.



*Şekil 9.2 Gözlemcinin görünümü (uçaktan)*

9.2. Koordinat Dönüşümleri ve Yön Kosinüsleri

İşte biraz matematik. Göründüğü kadar zor değil.

Bir referans çerçevesindeki bir nesnenin köşelerinin koordinatlarını biliyorsanız ve bunların bir diğerinde ne olduğunu bilmek istiyorsanız, bir koordinat dönüşümü yapmanız gerekir. (Diğer dönüşüm türünün geometrik dönüşüm olarak adlandırıldığını unutmayın; bu, nesnenin kendisi tek bir referans çerçevesi içinde hareket ettirildiğinde gerçekleşen şeydir). Bir noktanın dünya çerçevesinde (xw,yw,zw) koordinatları varsa, görünüm çerçevesinde (xv,yv,zv) koordinatları olacaktır. Bu nedenle Şekil 9.3'teki A noktasının dünya çerçevesinde (0,0,50) koordinatları ve görünüm çerçevesinde (0,-50,0) koordinatları vardır (ekranda görülenler daha sonra perspektif dönüşümü yoluyla hesaplanmalıdır). Dönmeler söz konusu olduğunda, bu iki koordinat kümesi arasında her zaman doğrusal bir ilişki vardır ve bu durum için genel terimlerle yazabiliriz:

xv = nll.xw + nl2.yw + nl3.zw

yv = n21.xw + n22.yw + n23.zw

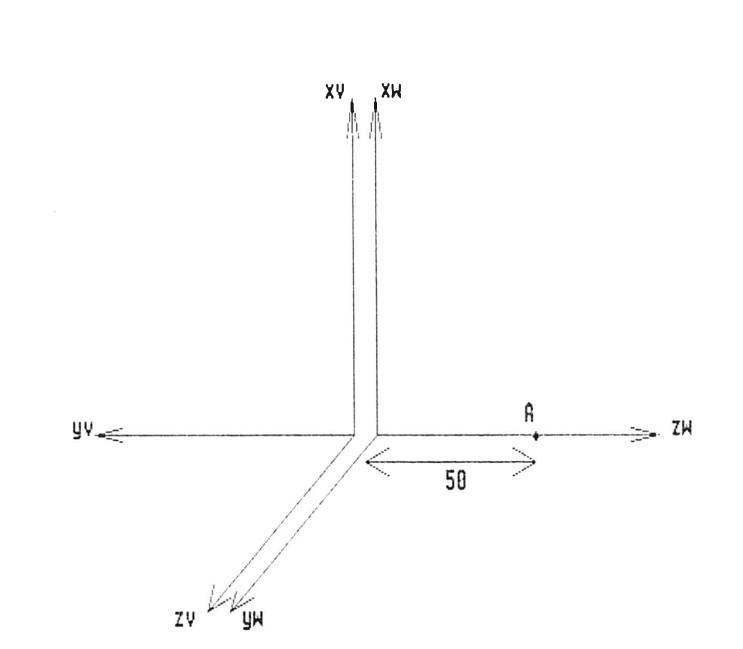
zv = n31.xw + n32.yw + n33.zw

burada n'ler çözülmesi gereken sayılardır. Bu ilişki bir matris çarpımı olarak da yazılabilir:

xv = n11 n12 n13 xw

yd = n21 n22 n23 \* yd

zv = n31 n32 n33 zw

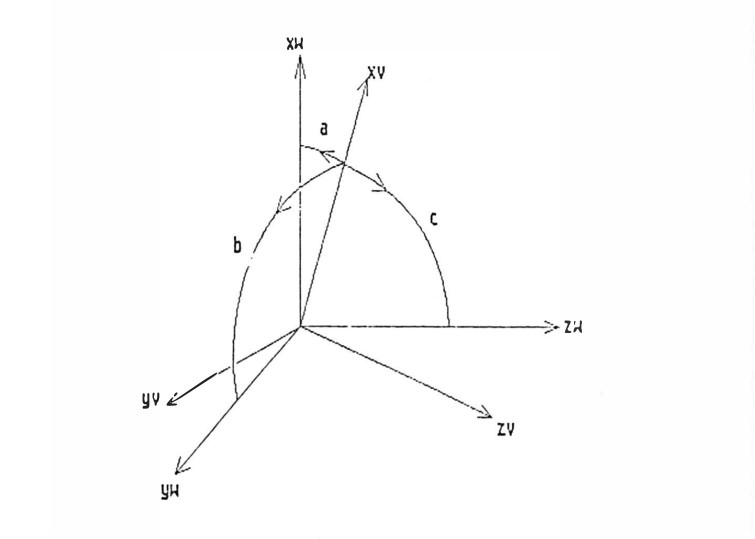


*Şekil 9.3 İki koordinat çerçevesinde görülen A noktası*

n matrisi dönüşüm matrisidir. nil, nl2 vb. elemanlar iki referans çerçevesinin göreli yönelimine özgüdür ve yön kosinüsleri olarak adlandırılır.

Yön kosinüslerinin geometriyle nasıl ilişkili olduğunu görmek için Şekil 9.4'e bakın. Yön kosinüsleri, referans çerçevelerinin eksenleri arasındaki açıların kosinüsleridir. Kafa karıştırıcı derecede karmaşık olmayan kapsamlı bir diyagram çizmek oldukça zordur ancak örneğin, nil vx ve xw arasındaki açının kosinüsüdür, nl2 xv ve yw arasındaki açının kosinüsüdür, nl3 xv ve zw arasındaki açının kosinüsüdür ve benzeri:

sıfır = cos(a), nl2 = cos(b), n!3 = cos(c).



*Şekil 9.4 Yönlü kosinüsler*

Eğer bu yön nesneleri bulunabilirse, dünya çerçeve koordinatlarını görüş çerçeve koordinatlarına dönüştürme sorunu çözülmüş olur. Ancak, joystick'in hareketlerini nesnelerin yönündeki değişikliklere dönüştürme sorunuyla karşı karşıyayız. Sorunu nesnelerin yönüne odaklanan bir stratejiyle çözmemiz gerektiği açıktır. İşte bunun yapılabileceği bir yol.

9.3. Taban Vektörleri ve Yön Kosinüsleri

Bir an için matematiği tamamen unutalım. Dünya çerçevesinde hareketsiz duran ve hem dünya çerçevesini hem de hareket eden gözlemciyi aynı anda görebilen ikinci, sabit, bağımsız bir gözlemcinin bakış açısından neler olduğunu görselleştirmeye çalışalım. Bu, yerde duran ve bir uçağın uçmasını izleyen bir adamın bakış açısıdır. Uçağı görüş çerçevesi olarak düşünün ancak gövdeyi zv ekseniyle, kanatları yv ekseniyle ve dikey kuyruk kanadını xv ekseni yönünde değiştirin. Uçakta olmasa da, sabit gözlemci uçağın herhangi bir andaki konumunu ve yönelimini biliyorsa pilota göre görüşü hesaplayabilir.

Örneğin, pilotun kumanda kolunu geri çektiğinde bu görünümün nasıl değişeceğini görmek için, uçağı yalnızca kanatların ekseni etrafında döndürmesi gerekir (açı, kumanda kolunun ne kadar geri çekildiğine bağlıdır), bu da yv ekseni etrafında bir dönüş demektir. Uçak dönüş sırasında ileri doğru hareket ettiğinden, bunun ek bir komplikasyonu da yukarı doğru uçmasıdır. Sabit gözlemci gibi, dünya etrafında dönerken görüş çerçevesinin konumunun ve yöneliminin sürekli bir kaydını tutmamız gerekir.

Bunu yapmak için, görüntü çerçevesi eksenlerinin yönlerinde üç birim vektör hayal edin. Vektör geometrisinde bu birim vektörlere özel bir ad verilir. Bunlara taban vektörleri denir. Programın en başında, görüntü çerçevesinin dünya çerçevesiyle çakışacak şekilde konumlandırıldığını varsayalım. Bu, uçağın havalandığı havaalanında ikinci bir dünya çerçevesi taban vektörü kümesine sahip olmaya eşdeğerdir. (Aslında bunların çakışacak şekilde başlaması gerçekten gerekli değildir ve genelde çakışmazlar, ancak argümanı görselleştirmeyi kolaylaştırır).

Şimdi, sonraki hareketin her aşamasında, görüntü çerçevesi birim vektörlerinin konumunu ve yönelimini kaydetmek gerekir. Uçağın sola (yaklaşık vx) veya yukarıya (yaklaşık vy) kaç derece döndüğüne dair sürekli bir toplam tutmak mümkün değildir, çünkü bu bilgiyi birçok hareketten sonra uçağın nihai yönelimine nasıl dönüştüreceğimizi bilmenin bir yolu yoktur. Önceki bölümde kullanılan Euler açıları yönteminde, ilk açı statik dünya çerçevesinin bir ekseni etrafındaki dönüşe atıfta bulunduğundan sürekli bir toplam tutmak mümkündü. Ancak şimdi, sürekli hareket eden görüntü çerçevesine atıfta bulunan açıları kullanıyoruz.

İşte büyük soru geliyor. Görüntü çerçevesi taban vektörlerinin konumlarının bir kaydını tutabildiğimizi varsayalım, bunların orijinal dönüşümle ne ilgisi var? Cevap çok basit: Görüntü çerçevesi taban vektörlerinin dünya çerçevesindeki bileşenler, dünya-görünüm dönüşüm matrisinin nil'den n33'e kadar olan elemanları olan yön kosinüsleridir. Başka bir deyişle, iv, jv ve kv görüntü çerçevesi taban vektörleri ve iw, jw ve kw dünya çerçevesi taban vektörleri olduğunda, aralarındaki ilişki şudur:

iv = n11.iw + n12.jw + n23.kw

jv = n21.iw + n22.jw + n23.kw

kv = n31.iw + n32.jw + n33.kw.

Veya, görünüm çerçevesi taban vektörlerini dünya çerçevesi bileşenleri açısından yazmak

n11 n21 n31

iv = n12, jv = n22, kv = n32

n13 n23 n33

Hareketin başlangıcında, görünüm çerçevesi ve dünya çerçevesi eksenleri hizalandığında, görünüm çerçevesi temel vektörleri bileşenlere sahipti

1 0 0

iv = 0, jv = 1, kv = 0

0 0 1

Eğer görüntü çerçevesi taban vektörlerinin bir kaydını tutabilirsek, kokpitten görüntüyü oluşturmak için hemen kullanılabilir yön nesnelerine sahip oluruz. Strateji basittir ancak yolda çözülmesi gereken bazı zorlu problemler vardır.

9.4. Taban Vektörlerinin Döndürülmesi: Keyfi Bir Eksen Etrafında Dönme

Görüntü çerçevesinin geçerli yönelimini sabitleyen temel vektörler, daha önce hangi hareketlerin gerçekleştiğine bağlıdır. Belirli bir anda görüntü çerçevesinin, Şekil 9.5'te gösterilen konumlarda temel vektörleriyle yönlendirildiğini varsayalım. vx ekseninin temel vektörü, iv, dünya çerçevesinde üç bileşene sahiptir: nil, nl2 ve nl3 (diğer birim vektörler jv ve kv'nin de bileşenleri vardır ancak açıklık sağlamak için bunlar diyagramda gösterilmemiştir). Şimdi, joystick'in vy ekseni etrafında bir dönüşe karşılık gelen bir hareketinin meydana geldiğini varsayalım. iv ve kv'nin yeni bileşenlerini bulmak için (jv bu dönüşte değişmeden kalır) bunları vy etrafında döndürmeliyiz. Vy ekseni, dönüş eksenidir ve dünya çerçevesinde yön kosinüsleriyle belirtilir. Ancak şanslıyız! Bu problem zaten çözülmüştür. Buna keyfi bir eksen etrafında dönüş denir. Bu noktada yv dünya çerçevesinde herhangi bir yeri işaret edebileceğinden, eksen çok keyfidir. Aslında problemin çözümü, bizim için en yararlı formatta verilmiştir. Yön kosinüsleri tarafından belirtilen bir eksen etrafında bir açıyla dönüş için bir matris biçimindedir. Tam da istediğimiz biçim. Dönüşüm, dünya çerçevesindeki herhangi bir başka eksen etrafında dönüş için de kullanılabilir. Gereken tek şey üç yön öğesidir.

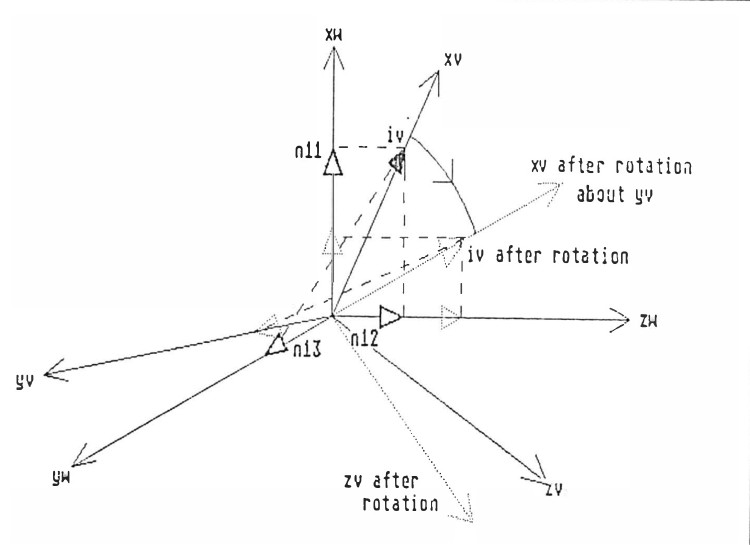
Bir kez oluşturulduktan sonra, dönme matrisi iv ve kv ile çarpılarak iv ve kv'nin yeni bileşenleri elde edilebilir; bu bileşenler daha sonra eski bileşenlerin yerini alır ve ayrıca doğrudan dünya-görünüm dönüşümünü oluşturmak için kullanılır (bir sorun var, bunu birazdan tartışacağız).

Yön kosinüsleri n1, n2 ve n3 olan bir eksen etrafında 𝛿 açısıyla dönüş için (kosinüsteki son indeks herhangi bir eksene atıfta bulunabileceğini gösterir), matris şu şekildedir:

n1.nl+(1-n1.n1)cos(𝛿) n1.n2(1-cos(𝛿))-n3sin(𝛿) n1.n3(1-cos(𝛿))+n2sini(𝛿)

n1.n2(1-cos(𝛿))+n3sin(𝛿) n2.n2+(1-n2.n2)cos(𝛿) n2.n3(1-cos(𝛿))-n1sin(𝛿)

n1.n3(1-cos(𝛿))-n2sin(𝛿) n2.n3(1-cos(𝛿))+n1sin(𝛿) n3.n3+(1-n3.n3)cos(𝛿)



*Şekil 9.5 Taban vektörünün dönüşü*

9.5. Hataların Biriktirilmesi

Genel olarak, joystick hareketleriyle kontrol edilen dünya çerçevesi üzerinden görüntü çerçevesini yönlendirmek için gereken tüm bileşenler yerindedir. Algoritmayı şu anki haliyle ortaya koyalım:

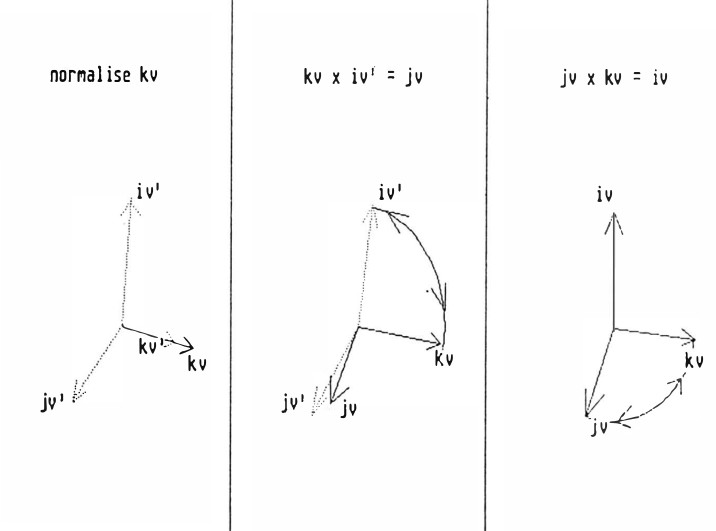
* joystick'in hareketi, görünüm birimi vektörlerinin görünüm çerçevesi eksenlerinden biri etrafında dönüşünü belirtir,
* diğer iki birim vektörü bu eksen etrafında döndürmek ve bunları yeni bileşenleriyle değiştirmek için dönüş matrisini oluşturun,
* dünya-görüntü dönüşümünü oluşturmak için artık yön kosinüsleri olarak adlandırılan birim vektörlerin bileşenlerini kullanın,
* dönüşümü gerçekleştir ve resmi görüntüle
* ve döngüyü tekrarlayın.

Bunların hepsi tamam ve işe yarıyor. Bir süreliğine.

Sonunda, biriken hatalar nedeniyle dejenere bir resme veya daha da kötüsü kaotik bir karmaşaya yol açacaktır. Şu haliyle programda yerleşik patolojik bir kendini yok etme özelliği vardır. Hesaplamalar tam sayı aritmetiğinde yapıldığından ve sinüsler ve kosinüsler 16384'te 1'den daha iyi olmayan bir doğrulukla hesaplandığından, yeterli dönüşüm sağlandığında, birim vektörlerde ve sonuç olarak dünya-görüntü dönüşümünde büyük hatalar birikecektir. Hayatta hiçbir şey mükemmel değildir ve bu da bu atasözünün iyi bir örneğidir. Ek olarak, algoritmanın joystick hareketlerinin, joystick hareketlerinden oluşturulan dönüşümden sırayla üretilen ekrandaki resme dayanarak yapıldığı yönünde bir geri bildirimi vardır. Bu, kaos yaratmak için gereken tüm bileşenlere sahiptir ve öyle de olur.

Hata birikimini yenmek için hata birikimi döngüsünün kırılması gerekir. Bu, her seferinde taban vektörlerinin yeniden oluşturulmasıyla elde edilir. Bu daha fazla çalışma gerektirir ancak sorunu çözer. Şekil 9.6, görünüm çerçevesi birim vektörlerinin yeniden oluşturulmasındaki aşamaları gösterir.

En önemli vektörler, hareket yönünü gösteren kv ve "yukarı" yönünü gösteren iv'dir. Bu ikisi olmadan, pilot açısından hareket yönünü veya hangi yönün yukarı olduğunu tanımlamak mümkün değildir. Son dönüşümdeki hatalar nedeniyle, biraz yanlış olan iv', jv' ve kv' olmak üzere üç birim vektörümüz olduğunu varsayalım. Hatalar, taban vektörlerinin birbirine dik olmamasına ve bire eşit boyuta sahip olmamasına neden olacaktır. İlk adım olarak, kv' vektörü normalleştirilir, yani büyüklüğü bire eşit hale getirilir. kv olur. Bu, en azından yönü biraz yanlışsa boyutunun da yanlış olmamasını sağlar. Biraz yanlış bir yönün tek etkisi, görünümün biraz hatalı olmasıdır, ancak görünüm zaten kumanda kolu tarafından sürekli olarak ayarlandığı için bu pek önemli değildir. İkinci olarak, her ikisine de 90 derecelik yeni bir vektör üretmek için kv ve iv'nin vektör çarpımı alınır. Bir vektör çarpım çarpımı tam olarak bu özelliğe sahiptir (bkz. Ek 5). Bu yeni vektör, hatada olmasaydı jv'nin sahip olacağı yöndedir. Daha sonra yeni vektör normalize edilir, yani büyüklüğü 1 yapılır ve yeni jv olur. Üçüncüsü, yeni kv ve yeni jv'nin vektör çarpım çarpımı alınır ve yeni bir iv üretmek için normalize edilir. Bu şekilde, üç birim vektör de her karede yeniden üretilir ve hatalar birikmez (örnek programdaki yeniden üretim aşamasını kaldırmak ve parçalanmanın gerçekleştiğini izlemek ilginçtir).



*Şekil 9.6 Temel vektörlerin yenilenmesi*

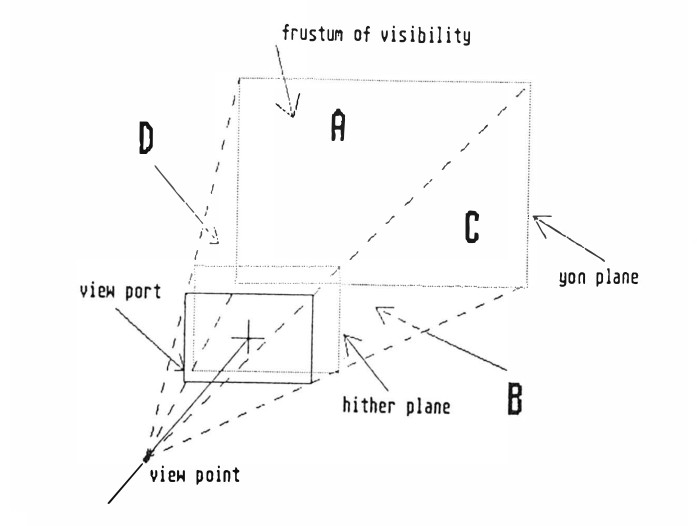
Yeni birim vektörlerin bileşenleri daha sonra görüntüleme dönüşüm matrisinin bileşenleri haline gelir ve döngü tekrarlanır.

Örnek programlarda görüldüğü gibi teknik detaylar ele alınmıştır.

9.6. 3D'de kırpma

Görüntü düzleminin (zv < 0) arkasında kalan bir nesnenin hiçbir parçası çizilmemelidir. Bu denenirse, program çökmez ancak ekranda görünen şey çöp olur. Bunun nedeni, çokgen çizim rutinlerinin köşe kenar listesinin çokgenin çevresi etrafında saat yönünde gitmesini beklemesidir ve bu, görüntü düzlemine geriye doğru yansıtılan çokgenler için yanlış olacaktır. Ayrıca, görüntü düzleminden çok uzakta bulunan nesneler de çizilmemelidir. Bunun nedeni, hiçbir şeyin bir pikselden daha küçük çizilememesi ve çok uzaktaki nesnelerin tutarsız bir piksel kümesine indirgenmesidir.

Bu bariz durumlar dışında, görüş alanının çok dışında kalan nesnelere zaman harcamanın bir anlamı yoktur. Bu görüş alanı, görüş noktasından görüş alanı sınırlarına kadar olan görüş hattı tarafından tanımlanan kesik piramit (kesik piramit) tarafından tanımlanır. Bu, Şekil 9.7'de gösterilmiştir.



*Şekil 9.7 3B'de Pencereleme*

Daha rahat bir uygulamada, çokgenlerin ekran penceresine kırpılma biçiminin 3B genellemesinde çokyüzlüleri kesik koninin sınırına kırpmak mümkün olurdu. Bu uygulamada bu çok zaman alıcı olurdu. Burada, simetri merkezi (Oox, Ooy, Ooz) görüş alanındaki nesneleri bulmak için kullanılır ve kırbaç açısı ekran sınırının ötesine uzanacak şekilde artırılır. Bu, görülemeyen uzak nesneleri çizerek biraz zaman harcanması anlamına gelir, ancak yakındaki nesneler merkezleri görüş alanının dışına geçtiği anda terk edilmez. Görünür olarak işaretlenirler, ancak ekran kırpmanın bir sonucu olarak ekranda yalnızca bir kısmı görünür.

Kesik koninin tepesi ve tabanı daha yüksek ve yon düzlemleri olarak adlandırılır. Örnek programda bunlar zv=100 (burada) ve zv=2000 (yon) denklemleriyle tanımlanır. Görüş alanının kesik koninin kenarları (görüntü alanı merkezinin görüş çerçevesi orijini ile çakıştığı yer) düzlemler tarafından tanımlanır

zv + 100 = xv A tarafı

zv + 100 = -xv B tarafı

(1.2).(zv + 100) = yv kenarı C

(1.2).(zv + 100) = -yv kenar D

ancak programda kullanılan gerçek taraflar yukarıda açıklanan nedenlerden dolayı bu sınırın ötesine uzanır ve aşağıdakiler tarafından tanımlanır:

8.(zv + 100) = ±xv kenar A ve B

8.(zv + 100) = ±yv kenarları C ve D

9.7. Gözlemcinin Hızı

Gözlemci (siz) dönüşleri yapmak için sadece kumanda kolunu kullanmaz. Gözlemcinin ayrıca zaman geçtikçe değişebilen bir hızı vardır. Hızı dahil etmek için yapılması gereken tek şey, gözlemcinin dünya çerçevesindeki konumunu hızla orantılı olarak artırmaktır. Hız bir vektördür, bu nedenle yönü ve boyutu vardır - hız, hızın büyüklüğüdür. Prosedür, gözlemcinin konumunun her bileşenini, her çerçeveyi, temel vektör kv'nin ilgili bileşeniyle çarpılan hıza orantılı bir miktarda değiştirmektir.

Başka bir deyişle, eğer görünüm çerçevesi sadece zw ekseninin yönünü gösteriyorsa, her seferinde sadece Ovz artırılmalıdır. Öte yandan, eğer görünüm çerçevesi xw ekseni boyunca gösteriyorsa, her seferinde sadece Ovx artırılmalıdır. Aradaki herhangi bir şey için, Ovx, Ovy ve Ovz, bu yönlerdeki kv bileşenleriyle orantılı olarak artırılmalıdır. Bu, gözlemcinin baktığı yönün hareket yönü olduğundan emin olmanızı sağlar. Ayrıntılar örnek programda açıklanmıştır.

9.8. Örnek Programlar

Bu programda A küpünün etrafında uçmak mümkündür. Program küpün ekranın ortasında ve gözlemcinin hareketsiz olduğu bir şekilde başlar. Basıldığında F2görüntü çerçevesinin sabit bir hızla kübe doğru hareket etmesi sağlanır (basıldığında F1geri çekilmesi sağlanır). Bundan sonra hareket joystick ile kontrol edilir. Küpün yanından uçmak ve ardından ona geri dönmek için bir dönüş yapmak mümkündür. 3D kırpma nedeniyle küp 100'den daha yakına gelirse veya 2000'den daha uzaksa veya görüş alanının dışındaysa görüntülenmez (yukarıya bakın). Hareket tuşuna basılarak durdurulabilir F6ve program tuşuna basılarak sonlandırılabilir F7.

9.8.1.wrld\_vw.s

Bu kontrol programıdır. Çoğu, önceki bölümdekine benzer. Joystick'in kontrolü altında uçurulabilir bir A küpü çizer. Bu sefer joystick, görüş çerçevesinin eksenleri etrafında, yani pilot etrafında dönüşler gerçekleştirir. Joystick geri çekildiğinde izleyici dünyaya yukarı bakar ve ileri hareket varsa yükselen bir yörünge izler. Joystick'in diğer hareketleri, izleyici dünya çerçevesinde uçuyormuş gibi karşılık gelen hareketi üretir. Bu şekilde bir nesnenin yanından uçmak ve sonra ona geri dönmek için bir yay boyunca taramak mümkündür.

Program yukarıda açıklanan sırayı takip eder. İlk olarak görüntü çerçevesi taban vektörleri başlatılır. Bunu takiben joystick okunur ve hemen dünya çerçevesindeki görüntü çerçevesi birim vektörleri döndürülür. Sonra hızın değişip değişmediğini görmek için klavye okunur. Bunu takiben dünya çerçevesindeki görüntü çerçevesinin yeni konumu hızdan ve artık yeni hareket yönüne işaret eden görüntü çerçevesi z ekseni taban vektörü kv'den hesaplanır. Düz bir çizgide olmayan harekette hız her zaman değişir (hız bir vektördür ve bu nedenle boyutu, hızı değişmese bile yönü değişse bile değişebilir). Son olarak birim vektörler hataların birikmesini önlemek için kendileri yeniden üretilir ve A küpünün resmini çizmeden önce doğrudan dünya-görüntü dönüşümünün elemanları olarak aktarılır.

Fonksiyon tuşları F1ve F2sırasıyla geri ve ileridir. F6durdurur ve F7sistemi sıfırlar. Kareler arasında klavye tamponu temizlenmediğinden tuşlara hafifçe basmaya ve basılı tutmamaya dikkat edin.

Mouon'da eylemsizlik gibi alt parçacıklar yoktur ancak bunlar önceki bölümde anlatılan doğrultuda dahil edilebilir.

9.8.2. çekirdek\_06.s

Tüm iş burada yapılır. Dircosinüs alt rutini taban vektörlerini yeniden üretir ve yeni değerleri görüntüleme dönüşüm matrisine geçirir. Yeniden üretmeyi yapmak için vektör çapraz çarpımları ve normalizasyon (yani vektörün boyutunu birliğe ölçekleme) gerekir. Bir vektörü normalleştirmek için bileşenlerinin her birini vektörün büyüklüğüne bölmek gerekir; bu da bileşenlerin karelerinin toplamının karekökü olarak hesaplanmalıdır. Bu, daha önce aydınlatma hesaplaması için kullanılan nrm vec rutini kullanılarak ele alınır.

Joy'daki alt yordamda, joystick okunur ve dünya çerçevesindeki bir eksen etrafında görüntü çerçevesi taban vektörlerini döndürmek için hemen eylem gerçekleştirilir; burada taban vektörlerden biri olan bu eksen, yön kosinüsleriyle tanımlanan herhangi bir eksen olabilir. Dönme matrisi v\_rot\_matx'te oluşturulur. Bunun elemanları oldukça büyüktür ancak satır ve sütun dizinleri yer değiştirmiş elemanların benzerliği nedeniyle bir seferde eleman çiftleri hesaplanarak genel iş en aza indirilir.

vel\_adj'de kv vektörünün işaret ettiği yön olan yeni hareket yönü, görüntü çerçevesinin yer değiştirmesini üretmek için hızla birleştirilir. Bu, basitçe kv'nin bileşenlerini hızla çarpıp bunları dünyadaki görüntü çerçevesi orijininin geçerli değeri olan Ovx, Ovy ve Ovz'ye eklemek anlamına gelir.

Nesnelerin görünürlüğü testi, yukarıda açıklanan ölçütleri izler; burada nesne çerçevesi kökeni (Oox,Ooy,Ooz) görüş alanı olarak tanımlanan kesikli alanda olup olmadığını görmek için incelenir. Bunu yapmak için, kökenin kendisi önce (Vox,Voy,Voz) haline geldiği görüş çerçevesine dönüştürülür.

Ekranın merkezini dünya çerçevesinin kökenine sıfırlamak için son bir rutin, sern adj dahil edilmiştir. Bu, dünya çerçevesindeki görünüm çerçevesini basitçe hareket ettirmekle aynı şey değildir çünkü perspektifin görünümünü etkiler. Görünüm çerçevesini ekranda merkezlemek, "uzayda uçma" deneyimlerine göre daha doğaldır.

9.8.3. bss\_06.s

Bu bölümde tanıtılan birkaç yeni değişkeni içerir: temel vektörler ve kumanda kolunun hareketinden kaynaklanan dönüşler.

\* wrld\_vw.s

\* Joystick control of the view frame for chapter 9

\*

\*SECTION TEXT

opt d+

bra main

include systm\_00.s screens and tables

include core\_06.s new subroutines

main bsr init allocate memory etc.

\* transfer all the data

bsr transfer

move.w oncoords,vncoords

move.w vncoords,wncoords

\* Initalise dynamical variables

move.w #0,Ovx view frame

move.w #0,Ovx starts off

move.w #-200,Ovz 200 behind world frame

\* Set up view frame base vectors

\* 1. iv

lea iv,a0 align

move.w #$4000,(a0) view

clr.w (a0)+ frame

clr.w (a0) axes

\* 2 jv

lea iv,a0 with

clr.w (a0)+ the

move.w #$4000,(a0)+ world

clr.w (a0) frame

\* 3. kv

lea jv,a0 axes

clr.w (a0)+

clr.w (a0)0

move.w #$4000,(a0)

clr.w speed start at rest

clr.w screenflag 0=screen 1 draw, 1=screen 2 draw

clr.w viewflag

loop4:

\* Switch the screens each time round

tst.w screenflag screen 1 or screen2?

beq screen\_1 draw on screen 1, display screen2

bsr drw2\_shw1 draw on screen 2, display screen1

clr.w screenflag and set the flag for next time

bra screen\_2

screen\_1:

bsr drw1\_shw2 draw on 1, display 2

move.w #1,screenflag

screen\_2:

\* Look for changes in the view frame angles

bsr in\_joy read joystick rotate view frame

\* See if the function keys have been pressed to change the speed

bsr key\_in

\* Adjust to new velocity

bsr vel\_adj

\* Recalculate view frame base vectors and set up the world-view

\* transform matrix

bsr dircosines

\* See if the object is within the visible angle of view

bsr viewtest

tst.b viewflag is it visible

beq loop4 no,try again

\* Construct compound objects from same face att different positions

move.w nparts,d7 how many parts in the object

subq #1,d7

lea inst\_angles,a0 list of angles for each part

lea ins\_disp,a1 ditto displacements

\* Do one face at a time

instance:

move.w d7,-(sp) save the count

move.w (a0)+,otheta next otheta

move.w (a0)+,ophi next ophi

move.w (a0)+,ogamma next ogamma

move.w (a1)+,Oox next displacements

move.w (a1)+,Ooy

move.w (a1),Ooz

movem.l a0/a1,-(sp) save position in the list

bsr otranvw object to world transform

bsr w\_tran\_v world to view transform

bsr illuminate if not hidden find the shade

bsr perspective perspective

bsr scrn\_adj centre window

bsr polydraw draw this face

movem.l (sp)+,a0/a1 restore pointers

move.w (sp)+,d7 restore the parts count

dbra d7,instance for all the parts of object

bra loop4 draw the next frame

\*SECTION DATA

include data\_00.s

include data\_03.s

include data\_05.s

\*SECTION WSS

include bss\_06.s

END

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Core\_06.s \*

\* subroutines for Chapter 9 \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

CORE6 EQU 1

include Core\_05.s

\* Find the direction cosines for the transform from the world frame to view frame.

\* These are components of the view frame base vectors in the world frame.

\* To avoid accumulating errors they are regenerated and normalised to a magnitude of:

\* 2^14.

dircosines

lea iv,a0

lea jv,a1

lea kv,a2

\* Kv is normalised

move.w (a2),d0

move.w 2(a2),d1

move.w 4(a2),d2

bsr nrm\_vec

move.w d0,(a2) new components

move.w d1,2(a2)

move.w d2,4(a2)

\* calc vj from cross product of vk & vi using subroutine AxB.

\* A pointer in a2: B pointer in a0:

bsr AxB

move.w d0,(a1)

move.w d1,2(a1)

move.w d2,4(a1)

\* finally the cross product of kv & jv is used for iv.

lea jv,a2

lea kv,a0

bsr AxB

lea iv,a1

move.w d0,(a1) regenerated iv

move.w d1,2(a1)

move.w d2,4(a1)

\* The components of the view frame base vectors in the world frame are the elements

\* of the transform matrix required for the world to view transform.

lea w\_vmatx,a0

lea iv,a1

lea jv,a2

lea kv,a3

move.w (a1)+,(a0)+ matrix elements of the view transform

move.w (a1)+,(a0)+

move.w (a1)+,(a0)+

move.w (a2)+,(a0)+

move.w (a2)+,(a0)+

move.w (a2)+,(a0)+

move.w (a3)+,(a0)+

move.w (a3)+,(a0)+

move.w (a3)+,(a0)+

rts

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

AxB

move.w 2(a2),d0 Ay

muls 4(a0),d0 bz\*Ay

move.w 4(a2),d1 Az

muls 2(a0),d1 By\*Az

sub.l d1,d0 Bz\*Ay-By\*Ax

\* 2nd component

move.w 4(a2),d1 Az

muls (a0),d1 Bx\*Az

move.w (a2),d2 Ax

muls 4(a0),d2 Bz\*Ax

sub.l d2,d1 Bx\*Az-Bz\*Ax

\* 3rd component

move.w (a2),d2 Ax

muls 2(a0),d2 By\*Ax

move.w 2(a2),d3 Ay

muls (a0),d3 Bx\*Ay

sub.l d3,d2 By\*Ax-Bx\*Ay

\* Reduce them to < word size by dividing by 2^14

move #14,d7

lsr.l d7,d0

lsr.l d7,d1

lsr.l d7,d2

\* normalise them

bsr nrm\_vec

rts

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Do a rotation of the view frame about one of the view frame axes in the world frame.

\* The direction cosines for the axis are the base vector components.

\* First a rotation about the view frame x-axis, vx.

rot\_vx

lea iv,a0 the axis of rotation

move.w vxangle,d1 the angle to rotate

bsr v\_rot\_matx construct the rotation matrix

\* only jv and kv are affected

lea jv,a0 1st transform

bsr rot\_view

lea kv,a0 2nd transform

bsr rot\_view

rts

\*--------------------------------------------------

rot\_vy

lea jv,a0

move.w vyangle,d1

bsr v\_rot\_matx

\* only iv and kv are affected

lea iv,a0 1st transform

bsr rot\_view

lea kv,a0 2nd transform

bsr rot\_view

rts

\*--------------------------------------------------

rot\_vz

lea kv,a0

move.w vzangle,d1

bsr v\_rot\_matx

\* only iv and kv are affected

lea iv,a0 1st transform

bsr rot\_view

lea jv,a0 2nd transform

bsr rot\_view

rts

\*--------------------------------------------------

\* Rotate a view frame base vector. The vector is pointed to by a0. Since it is

\* a unit vector it is specified by three components which are the direction cosines.

\* (nx, ny, nz).

rot\_view

moveq #2,d6 rows in matrix

lea vrot\_matx,a3

link a6,#-6

rot\_vw1

move.w (a0),d0 nx components

move.w 2(a0),d1 ny

move.w 4(a0),d2 nz

muls (a3)+,d0 nx\*Mi1

muls (a3)+,d1 ny\*Mi2

muls (a3)+,d2 nz\*Mi3

add.l d1,d0

add.l d2,d0

lsl.l #2,d0

swap d0

move.w d0,-(a6)

dbra d6,rot\_vw1

move.w (a6)+,4(a0) z

move.w (a6)+,2(a0) y

move.w (a6)+,(a0) x

unlk a6

rts

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Construct the rotation matrix for rotations about an arbitrary axis specified by a

\* unit vector with components (direction cosines) n1, n2, n3.

\* ENTRY: Pointer to direction cosines in a0: Angle in d0.

v\_rot\_matx

lea vrot\_matx,a6

bsr sincos

move.w d2,d6 sine delta

move.w d3,d7 cos delta

\* elements M12 and M21

move #16384,d5

move d5,d0

move.w (a0),d1 n1

muls 2(a0),d1 n1\*n2

lsl.l #2,d1

swap d1

sub.w d7,d0 1-cosdelta

move d0,d4

muls d1,d0

lsl.l #2,d0

swap d0 n1\*n2(1-cosdelta)

move d0,d2

move.w 4(a0),d1 n3

muls d6,d1 n3\*sindelta

lsl.l #2,d1

swap d1

sub.w d1,d0 n1\*n2(1-cosdelta)-n3\*sindelta

move.w d0,2(a6) M12

add.w d1,d2 n1\*n2(1-cosdelta)+n3\*sindelta

move.w d2,6(a6) M21

\* elements M13 and M31

move d4,d0 1-cosdelta

muls (a0),d0 n1\*(1-cosdelta)

lsl.l d0

swap d0

muls 4(a0),d0 n1\*n3(1-cosdelta)

lsl.l #2,d0

swap d0

move d0,d2

move.w 2(a0),d1 n2

muls d6,d1 n2\*sindelta

lsl.l #2,d1

swap d1

add.w d1,d0 n1\*n3(1-cosdelta)+n2\*sindelta

move.w d0,4(a6) M13

sub.w d1,d2 n1\*n3(1-cosdelta)-n2\*sindelta

move.w d2,12(a6) M31

\* elements M23 and M32

move d4,d0 1-cosdelta

muls 2(a0),d0 n2\*(1-cosdelta)

lsl.l #2,d0

swap d0

muls 4(a0),d0 n2\*n3(1-cosdelta)

lsl.l #2,d0

swap d0

move d0,d2

move.w (a0),d1 n1

muls d6,d1 n1\*sindelta

lsl.l #2,d1

swap d1

sub.w d1,d0 n2\*n3(1-cosdelta)-n1\*sindelta

move.w d0,10(a6) M23

add.w d1,d2 n2\*n3(1-cosdelta)+n1\*sindelta

move.w d2,14(a6) M32

\* elemnt M11

move.w (a0),d1 n1

muls d1,d1 n1\*n1

lsl.l #2,d1

swap d1

move d5,d2 1

sub.w d1,d2 1-n1\*n1

muls d7,d2 (1-n1\*n1)cosdelta

lsl.l #2,d2

swap d2

add.w d2,d1 n1\*n1+(1-n1\*n1)cosdelta

move.w d1,(a6) M11

\* element M22

move.w 2(a0),d1 n2

muls d1,d1 n2\*n2

lsl.l #2,d1

swap d1

move d5,d2 1

sub.w d1,d2 1-n2\*n2

muls d7,d2 (1-n2\*n2)cosdelta

lsl.l #2,d2

swap d2

add.w d2,d1 n2\*n2+(1-n2\*n2)cosdelta

move.w d1,8(a6) M22

\* element M33

move.w 4(a0),d1 n3

muls d1,d1 n3\*n3

lsl.l #2,d1

swap d1

move d5,d2

sub.w d1,d2 1-n3\*n3

muls d7,d2 (1-n3\*n3)cosdelta

lsl.l #2,d2

swap d2

add.w d2,d1 n3\*n3+(1-n3\*n3)cosdelta

move.w d1,16(a6) M33

rts

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

w\_tran\_v

move.w wncoords,d7

ext.l d7 any to do?

beq w\_tranv3

subq.w #1,d7

lea wcoordsx,a0

lea wcoordsy,a1

lea wcoordsz,a2

lea vcoordsx,a3

lea vcoordsy,a4

lea vcoordsz,a5

exg a3,d3 save cos we're short of registers

link a6,#-6 save 3 words

w\_tranv1

moveq.l #2,d6 3 rows in matrix

lea w\_vmatx,a3 init max pointer

\* calculate the next vx, vy and vz

w\_tranv2

move.w (a0),d0 wx

move.w (a1),d1 wy

move.w (a2),d2 wz

sub.w Ovx,d0

sub.w Ovy,d1

sub.w Ovz,d2

muls (a3)+,d0 wx\*Mi1

muls (a3)+,d1 wy\*Mi2

muls (a3)+,d2 wz\*Mi3

add.l d1,d0

add.l d2,d0 wx\*Mi+wy\*Mi2+wz\*Mi3

lsl.l #2,d0

swap d0

move.w d0,-(a6)

dbra d6,w\_tranv2 repeat for 3 elements

move.w (a6)+,(a5)+

move.w (a6)+,(a4)+

exg a3,d3 restore vx, save matx pointer

move.w (a6)+,(a3)+

exg a3,d3 save vx, restore matx pointer

addq.l #2,a0 point to next wx

addq.l #2,a1 wy

addq.l #2,a2 wz

dbra d7,w\_tranv1 repeat for all ocoords

unlk a6 close frame

w\_tranv3

rts

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Set the velocity components

vel\_adj

lea kv,a0

moveq.l #14,d7 ready to divide by 2^14

move.w speed,d0

lsl.w #3,d0 scale it

move d0,d1

move d0,d2

muls (a0),d0 v\*VZx

lsr.l d7,d0 /2^14

add.w d0,Ovx xw speed component

muls 2(a0),d1 v\*VZy

lsr.l d7,d1

add.w d1,Ovy zw speed component

muls 4(a0),d2 v\*VZz

lsr.l d7,d2

add.w d2,Ovz

rts

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* test whether the primitive is vsible. see whether its centre (oox,Ooy,Ooz) lies within

\* the angle of visibilty. Oox, Ooy and Ooz are transformed to view coords and then tested.

viewtest

moveq.l #2,d6 rows in matrix

lea w\_vmatx,a3

link a6,#-6

move.w Oox,d3

addi.w #50,d3

move.w Ooy,d4

addi.w #50,d4

move.w Ooz,d5

addi.w #50,d5

sub.w Ovx,d3 Oox-Ovx relative to the view frame

sub.w Ovy,d4

sub.w Ovz,d5

tran0v

move d3,d0

move d4,d1

move d5,d2

muls (a3)+,d0 \*Mi1

muls (a3)+,d1 \*Mi2

muls (a3)+,d2 \*Mi3

add.l d1,d0

add.l d2,d0 \*Mi1+\*Mi2+\*Mi3

lsl.l #2,d0

swap d0

move.w d0,-(a6)

dbra d6,tran0v repeat for three elements

move.w (a6)+,d3 Voz

move.w (a6)+,d2 Voy

move.w (a6)+,d1 Vox

move.w d3,Voz

move.w d2,Voy

move.w d1,Vox

unlk a6

\* Clip Ovz. For visibility must have 100<Voz<2000

cmpi.w #100,d3 test(Voz-100)

bmi invis

cmpi.w #2000,d3 test(Voz-2000)

bpl invis

\* is it within the view angle?

addi.w #100,d3 Voz+100

add.w d3,d3 \*2

add.w d3,d3 \*4

add.w d3,d3 \*8

\* First test horizontal position

tst.w d2 is Voy +ve or -ve

bpl pos\_y

neg.w d2

pos\_y

cmp.w d2,d3 Voy is +, (test(8\*(Voz+100)\_Voy))

bmi invis

\* Test vertical position

tst.w d1 Vox

bpl pos\_x

neg.w d1

pos\_x

cmp.w d1,d3 test(8(Voz+100)-Vox)

bmi invis

\* It IS visible

st viewflag

rts

\* It is INVISIBLE

invis

sf viewflag

rts

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*Adjust screen coords so that view frame (0,0) is at centre

scrn\_adj

move.w vncoords,d7

beq adj\_end

subq.w #1,d7

lea scoordsy,a0

adj\_loop

subi.w #100,(a0)+

dbra d7,adj\_loop

adj\_end

rts

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* bss\_06.s \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

include bss\_05.s

\* VARIABLES FOR ROTATING THE VIEW FRAME

iv ds.w 3 view frame base vector components in world

jv ds.w 3

kv ds.w 3

vxangle ds.w 1 rotation angles about these axes

vyangle ds.w 1

vzangle ds.w 1

vrot\_matx ds.w 9 rotation matrix about an arbitrary axis

\* VISIBILTY

viewflag ds.w 1

Vox ds.w 1 object centre in view frame

Voy ds.w 1

Voz ds.w 1

10. Bir Dünya Sahnesi

Bu bölümde çok sayıda nesnenin yer aldığı bir dünya inşa ediliyor.

Tek bir grafik ilkelinden birkaç tane içeren bir sahneye geçiş, bir sürü yeni sorun getirir. Örneğin, birçok nesnenin olduğu karmaşık bir sahnede, mekansal ilişkiler korunmalıdır; ön plandaki nesneler uzaktaki nesneler tarafından gizlenmemelidir. Nesneleri gözlemciden uzaklıklarına göre çizim için sıralayan bir tür derinlik sıralaması gereklidir.

Gözlemcinin yakın çevresinin dışındaki tüm nesneleri görmezden gelmek için sağlam bir strateji de aynı derecede önemlidir. Bir manzaraya yayılmış yüzlerce nesneden oluşan bir dünyada, hepsini çizmeye çalışmak anlamsızca zaman alıcı olurdu. Gerçek hayatta olduğu gibi, gözlemcinin yalnızca yakınlarda olan ve mevcut kararları etkileyenlerle ilgilenmesi gerekir. Çok nesneli dünyanın bu yönlerini sırayla inceliyoruz.

10.1. Bir Veritabanı

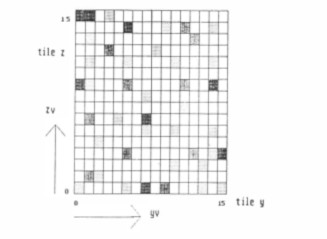
Karmaşık dünyadaki her nesneyle ilişkilendirilen, onun niteliklerinin bir listesi (tür, konum, renk, dönüş açıları, vb.) olacak ve tüm nesnelerin listelerinin kümesi bir veritabanıdır. Gözlemcinin gördüğü görünümü çizmek için gereken tüm bilgileri içerir. Bu veritabanının bellekte tam olarak nasıl düzenlendiği, grafikler için erişilebileceği hızı belirlemede çok önemlidir.

Bu noktayı daha iyi açıklamak için, veritabanındaki nesneleri sıralamada mevcut seçenekleri göz önünde bulundurun. Nesneler, veritabanına artan x (dünya) koordinatları veya artan y koordinatları veya artan z koordinatları sırasına göre veya aslında hiçbir mekansal sıra olmaksızın rastgele girilebilir. Nesneler, türlerine, renklerine veya herhangi bir niteliklerine göre listelenebilir. Tüm olasılıklar arasında, çizilecek nesnelere, yani gözlemcinin hemen yakınında olanlara hızlı erişim sağlayanlar olacaktır. Bunu başarmak için bir tür konum sıralamasına ihtiyaç duyulduğu açıktır.

10.1.1. Bir Harita

Bir nesnenin dünyadaki konumu, (xw,yw,zw) biçimindeki üç koordinatıyla belirtilir. Veritabanını tek bir koordinatta (xw veya yw veya zw) sıralamanın, her nesnenin komşularına göre nerede olduğuna dair anında bir resim sağlamayacağı açıktır.

Gerekli olan, nesnelerin 3B düzende düzenlendiği bir veritabanıdır. Tanımlanan şeyin bir haritadan başka bir şey olmadığı anlaşılana kadar bunu görselleştirmek zordur. Sıradan bir rota haritasına olan benzerlik, tıpkı Dünya'nın yüzeyi gibi bir yüzeyde oturan nesnelerden oluşan inşa edeceğimiz dünya için oldukça kesindir. Bu tür bir haritanın (2B dizidir) avantajı, belirli bir bölgede bulunan tüm nesnelerin mekansal ilişkilerinde hemen belirgin olmasıdır.



*Şekil 10.1 Dünya 'fayanslarının' düzeni*

Aslında yapılan şey Şekil 10.1'de gösterilmiştir. Dünya uzayı, her biri 256\*256 boyutlarında olan 16\*16'lık bir "fayans" dizisine bölünmüştür (tıpkı banyo duvarındaki gibi). Her fayans, görüntüleme için dikkate alınacak bir uzay birimidir. Bir nesne koleksiyonu içerebilir; örnek programda, basitlik açısından yalnızca bir tane içerir. Elbette bu çok geniş bir dünya değildir, ancak yöntemde onu bu boyutlarla sınırlayan hiçbir şey yoktur; istediğiniz kadar büyük ve tek tek fayanslar istediğiniz kadar küçük olabilir. Ancak, "sarma", gözlemci herhangi bir kenardan saptığında karşı tarafta yeniden belirecek şekilde gerçekleşir; bu şekilde dünya, bir küre gibi etkili bir şekilde "sonsuzdur". Bizim amaçlarımız için, yöntemi göstermek için 16\*16'lık bir fayans dünyası yeterlidir. Her fayans, görüntüleme amaçları için tek bir varlık olan bir uzay bölgesini tanımlar. Gözlemcinin gördüğü görüntüyü oluşturmak için yapılması gereken tek şey, kişinin fayans ızgarasındaki konumunu bulması, en yakın komşu fayansları seçmesi, hangilerinin gözlemcinin önünde olduğunu bulması ve üzerlerine yerleştirilen nesneleri çizmesidir.

Bu 2B dizi ID bitişik RAM'de nasıl düzenlenebilir? Burada yeni bir şey yok. Ekranın kendisi, bellekte bir ID veritabanı olarak temsil edilen 2B bir dünyadır. Piksel bir döşemeye benzer ve rengini belirten dört bit, döşemedeki nesnenin niteliklerini belirten veri listesine benzer. Her bir öğenin bitişik olanlara bağlandığı bu şekilde bir bilgi düzenlemesine bağlı liste denir. Bu durumda, bağlantılar kalıcıdır ve dizideki fiziksel konum tarafından ima edilir. Dünya veritabanı, bu nedenle 16\*16 döşeme dünyasındaki bir döşemenin niteliklerini tutan her biri 256 baytlık bir listedir. Örnek programda data\_08.s dosyasında tutulur. Liste map\_base'de başlar ve her 16. bayt z yönünde yeni bir döşeme başlatır. Listedeki döşeme konumu, mod 16, 16 y değerini temsil eder. Bu modelde dünya düzdür ve x değişmez.

Nitelikler için çok az bilgiye ihtiyaç vardır, çünkü uzaydaki konum otomatik olarak listedeki kiremitin konumuna dahil edilir. İlk nibble arka planın rengini (1-15) verir ve ikincisi kiremitin üzerinde duracak nesnenin türünü verir. Şu anda yalnızca altısı mümkündür (data\_06.s'de listelenmiştir), ancak prensipte bir sınır yoktur.

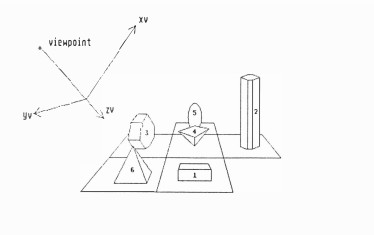
10.2. Sıralama

Yukarıda belirtildiği gibi, gözlemciye yakın mesafedeki görünür nesneler tanımlandıktan sonra, daha uzaktakilerin önce çizilmesi için bunları çizim sırasına koyma sorunu vardır. Bu, genellikle ressamın algoritması olarak bilinir, çünkü bir resmi boyarken son fırça darbesi öncekilerin üzerine biner.

Verileri sıraya koymak için birçok iyi bilinen algoritma vardır. Daha egzotik çeşitlerin çoğu, çok sayıda giriş (kayıt) içeren büyük veritabanlarını ele almak için geliştirilmiştir. Bizim durumumuzda, az sayıda (<16) kaydı derinlemesine sıralamak gerekir. Bu düzeyde sıralama, kabarcık sıralaması adı verilen en basit sıralama yöntemlerinden biriyle verimli bir şekilde yapılır. Bu aşamada, kayıt olarak çizilecek nesneler hakkında niteliklere ve diğer birikmiş verilere atıfta bulunduğumuzu unutmayın. Bir kayıt, her veri türünün belirli bölümlere veya "alanlara" sınırlandırıldığı farklı türlerdeki bir veri kümesidir. Örnek programlarda görünür nesneler için veriler bu şekilde taşınır. Karoya çizilecek tüm ilgili verileri içeren bir kayıt oluşturulur ve derinlemesine sıralama sırasında kayıtlar aslında bir deste kart gibi sıralanır. Bu şekilde, derinlik alanı sıralama için temel olsa da, çizim için başka bilgiler de taşır ve daha sonraki bir aşamada ek verilerin alınmasını en aza indirir. Elbette, işleri çok yavaşlatmamak için kaydı kısa tutmak önemlidir. Örnek programda bir kayıt 7 alana bölünmüş 2 uzun kelimeden oluşmaktadır.

10.2.1. Bir Kabarcık Sıralaması

Kabarcık sıralamasını programdan doğrudan bir örnekle açıklayalım. Burada, görüntülenecek görünür nesneler için kısa bir kayıt listemiz var. Sıralamanın dayandığı alan, kayıttaki ikinci kelimedir. Nesnenin, pozitif z yönündeki görüntü çerçevesinin kökenine olan uzaklığıdır, yani izleyicinin baktığı yöndür. Diğer alanlar sıralama için önemsizdir. Şekil 10.2, görüntü çerçevesinin önündeki basit nesnelerin olası bir düzenlemesini gösterir. Her nesnenin üzerindeki sayı, türü, yani kaydındaki ikinci Alanın içeriğidir. Arkadaki nesnelerin öndeki nesnelerin arkasında kalması için çizilmeleri gereken uygun sıra şudur: 2,1,4,5,6,3. Ancak bunun, döşemelerin veritabanından alındığı sıra olması olası değildir. 6,1,3,4,2,5 sırasıyla geri çekildiklerini varsayalım. Sıralama şimdi başlıyor.



*Şekil 10.2: Nesnelerin derinlik sıralaması*

Kabarcık sıralamasındaki prosedür, her girdiyi halefiyle karşılaştırarak listeyi incelemek ve gerekirse bir geçiş yapmaktır. Mevcut durumda, listeyi listenin en üstüne ilk çizilecek nesnelerle sıralayacağız, yani liste şu sırayla olacaktır: uzak nesneler - yakın nesneler. İlk taramada, önce ilk çift 6 ve 1 incelenir, yanlış sırada oldukları bulunur ve değiştirilir. Aynı zamanda, listenin sırasız olduğunu kaydetmek için bir bayrak ayarlanır. Bu, 1'i ilk girdi, 6'yı ikinci girdi olarak bırakır. Sonra bir sonraki çift 6 ve 3 incelenir. Buradaki sıra TAMAM olduğundan hiçbir geçiş yapılmaz. Bu, tüm liste boyunca devam eder. Her geçiş yapıldığında bayrak ayarlanır (elbette yalnızca bir kez ayarlanabilir, bu nedenle sonraki değişimler bayrağa hiçbir şey yapmaz). Aşağıdaki satırlar ilk sıralamanın ilerleyişini göstermektedir:

6.1.3.4.2.5 başlangıç

1.6.3.4.2.5 Test edilen 1. çift

1.6.3.4.2.5 2.

1.6.4.3.2.5 3.

1.6.4.2.3.5 4.

1,6,4,2,5,3 5.

Uzaktaki nesnelerin baloncuklar gibi nasıl yukarı doğru "yüzdüğüne" dikkat edin.

Listenin sonunda bayrak bir geçiş yapılıp yapılmadığını görmek için test edilir. Eğer yapıldıysa tüm liste tekrar test edilir. Bu, bayrağın ayarlanmadığı bir geçiş yapılana kadar tekrarlanır, bu durumda sıralı liste ve sıralama tamamlanmış sayılır.

10.2.2. Görüntüleme Dönüşümü

Bu bölümde gözlemcinin gördüğü görünümü oluşturmanın iki farklı yolunu ekliyoruz. İlki kontrol matrislerini kullanır ve önceki bölümde kullanılan görünüm dönüşümünün daha basit bir versiyonudur. İkincisi tamamen farklıdır ve çok daha basittir; Bölüm 8'de karşılaşılan Euler açılarını kullanır ve temel uçuş simülatörlerinde yaygın olarak kullanılır. Açıların tanımlanma şeklinin bir sonucu olarak biraz sınırlıdır. Önce kontrol matrislerinin uygulamasını tartışıyoruz.

10.2.3. Kontrol Matrisleri

Tüm dönüşümlerin bir gözlemcinin bakış açısından bir sahneyi sunmak için yapıldığı aşamaya ulaştığımızı varsayalım. Tüm görünür nesnelerin köşeleri daha sonra gözlemcinin referans çerçevesinde, yani görüş çerçevesinde verilecektir. Örneğin, joystick'in bir hareketinin sonucu olarak gözlemci başını sola hareket ettirirse, yeni görüşü göstermek için gereken tek şey köşeleri sağa döndürmektir. Gözlemcinin referans çerçevesindeki herhangi bir eksen etrafında dönmesi, görüş çerçevesi köşe koordinatlarını ters yönde döndürerek uygulanabilir. Böyle bir dönüşüme koordinat dönüşümü denir çünkü farklı bir koordinat sisteminden, yani gözlemcinin döndürülmüş koordinat sisteminden görülen görüşü hesaplar.

Yani öyle görünüyor ki gözlemcinin dünyada uçarkenki görüşünü göstermek için gereken tek şey, görüş karesi koordinatlarını bugüne kadarki toplam hareketini temsil eden dönüş matrisleri dizisiyle çarpmaktır. İşe yaramayacak! Önce toplam dönüş dizisinin bir kaydı tutulmalı ve sonra her kare için sırayla çarpılmalıdır. Hızlı grafikler için tam olarak verimli bir algoritma değil. Bir süre sonra resim, her kare için yüzlerce matris çarpımı yapıldığından tamamen durur. Çözüm nedir?

Bu problemin çözümü, önceki bölümde kullanılan, görüntü çerçevesi taban vektörlerinin döndürüldüğü ve daha sonra görüntü dönüşümünü oluşturmak için kullanıldığı yönteme çok benzerdir. Bu durumda prosedür geriye doğru yapılır. Herhangi bir anda, önceki kareyi görüntülemek için yapılan hesaplamaların bir sonucu olarak, görüntü dönüşüm matrisini biliriz. Bu, bir sonraki karenin başlangıç ​​noktasıdır. Hesaplamaların sonunda olayların sırası şöyle olacaktır: 1) köşeleri görüntü çerçevesine dönüştürmek için görüntü dönüşümünü yapmak, 2) bahsettiğimiz görüntü çerçevesi eksenleri etrafında döndürmeleri yapmak, 3) son olarak perspektif dönüşümünü yapmak ve ardından gelen her şeyi yapmak. İşte şimdi problemin çözümü. Görüntü dönüşümünü (V) ve görüntü çerçevesi dönüşlerini (©) ayrı dönüşümler olarak düşünmek yerine, önce görüntü çerçevesi köşelerini (PV) ve sonra döndürülmüş köşeleri (PV') üretmek için dünya çerçevesinde sırayla köşelere (PW) yapılması gerekir.

(C)(V)(PW) = (V')(PW) = (PV').

(V') döndürülmüş bir görünüm dönüşümü üretmek için önceden © ve (V)'yi ayrı ayrı birleştiriyoruz (çarpıyoruz)

(C)(V)(PW) = (C)(PV) = (PV'),

Bu şemada gözlemcinin her dönüşü, görünüm dönüşümünün dönüşe uygun bir "kontrol" matrisiyle önceden çarpılmasıyla sağlanır. Görünüm çerçevesi xv, yv ve zv eksenleri etrafındaki ayrı dönüşler için kontrol matrisleri şunlardır:

| 1 0 0 |

| |

(Cx) = | 0 cosθ sinθ |

| |

| 0 -sinθ cosθ |

| |

| cosθ0 -sinθ |

| |

(Cy) = | 0 1 0 |

| |

| sinθ 0 cosθ |

| |

| rahat siny 0 |

| |

(Cz) = | -siny rahat 0 |

| |

| 0 0 1 |

| |

Bunların Bölüm 6'daki geometrik dönüşümlerle tamamen aynı olduğunu, ancak sinüs terimlerinin ters işarete sahip olduğunu unutmayın. Bunun nedeni

günah(-θ) = -sin(θ)

ve koordinat dönüşümlerinin negatif açılı geometrik dönüşümlerle aynı olduğunu, yani geriye doğru dönüşlere karşılık geldiğini gösterir. Bu, matematiksel olarak doğru olduğunu bildiğimiz şeyi söyler: gözlemcinin başını sola döndürmek, sahneyi sağa döndürmekle aynı sonucu elde eder. (Ek 6'ya bakın).

Dönüşlere karşılık gelen fiziksel hareketler Şekil 10.3'te gösterilmiştir. Bunlar: sapma (x ekseni etrafında dönüş), eğim (y ekseni etrafında dönüş) ve yuvarlanmadır (z ekseni etrafında dönüş).

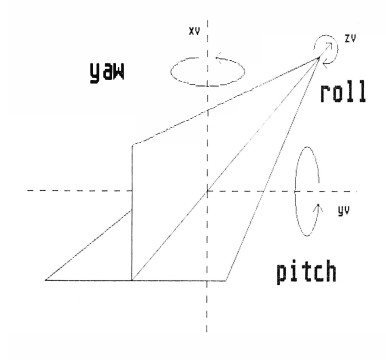
İşleri hızlandırmak için kontrol matrisleri önceden hesaplanabilir. Dönmelerin her zaman 1 derecelik artışlarla gerçekleştiği kabul edilirse, matrislerin elemanları sine(l) ve cos(l) olacaktır (her zamanki gibi 16384 ile çarpılır). Bu, açı artışlarının 5 olarak alındığı örnek program dosyası dat\_07.s'de yapılan şeydir, ancak burada dönmeler yalnızca xv ve yv eksenleri etrafında gerçekleşir.

Hataların birikmemesini sağlama ihtiyacı hala devam ediyor. Dolayısıyla, görünüm dönüşümünün satırlarının görünüm çerçevesi taban vektörleri olarak görselleştirilebileceğini hatırlayarak, 9. Bölüm'de yapıldığı gibi görünüm matrisi satırlarını vektör ürünlerine göre yeniden oluşturuyoruz.

Tüm bu aşamaların detayları wrld\_scn örnek programında gösterilmiştir.

10.2.4. Euler Açıları

Bunları 8.1.1. bölümünde zaten tartıştık. Euler açıları, yalnızca üç açı kullanarak bir referans çerçevesinin diğerine göre yönelimini belirtmenin bir yoludur, ancak açıların nasıl tanımlandığına ilişkin bazı kısıtlamalar vardır. En önemlisi, farklı eksenler etrafındaki dönüşleri sabit bir sırayla belirtmeleridir. Birçok olası kombinasyon vardır. Aşağıda tanımlanan dizi, havacılık mühendislerinin sevdiği bir dizidir ve 321 dizisi olarak adlandırılır, çünkü x, y ve z eksenleri etrafındaki dönüşleri sırayla tanımlar. Bunlar, kumanda kolunun hareketleriyle ilişkilidir ve bu nedenle sapmayı (yatağı), eğimi ve yuvarlanmayı tanımlar, ancak burada sapmanın, dünya çerçeve ekseni wx etrafındaki bir başlangıç ​​dönüşü olması nedeniyle, 10.3.1. bölümünde açıklanandan farklı olduğunu unutmayın. Gözlemcinin fiziksel dönüşleri Şekil 10.3'te gösterilmiştir.



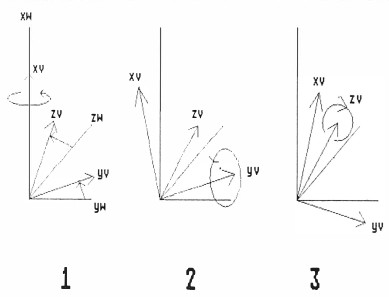
*Şekil 10.3 Görünüm çerçevesi dönüşleri için havacılık terimleri*

İşte dünya referans çerçevesini gözlemcinin görüş çerçevesine taşıyan dönüşlerin sırası (yer değiştirmeler zaten çıkarılmıştır). Şekil 10.4'te gösterilmiştir. Her iki çerçeve de başlangıçta çakışıktır ve dönüşler, o anda nerede olurlarsa olsunlar, görüş çerçevesi eksenleri etrafında yay çizer:

1. x ekseni etrafında 0 döndür - her iki kare için de aynı (sapma)
2. y ekseni etrafında 4> döndür (eğim)
3. z ekseni etrafında y ile döndür (yuvarlan)

Son ürün, görüş çerçevesinin yönelimidir.

Bölüm 8.1.1'e geri bakıldığında, bunun tam olarak orada yapılan dönüşlerin dizisi olduğu ve bu nedenle sonuçların, özellikle son matris ürününün doğrudan kullanılabileceği görülecektir. Sonuçlar eulr\_sen örnek programında gösterilmiştir.



*Şekil 10.4 Euler açılarının dönüş dizisi*

10.3. Çalışma Süreleri

Bu bölümdeki örnek program, ilkel bir uçuş simülatöründe olduğu gibi, joystick'in kontrolü altında 256 farklı grafik varlığı içeren bir dünyada dolaşmanıza olanak tanır. Burada hiçbir sınırlama yoktur; ek zaman cezası olmadan daha büyük bir dünya veritabanı oluşturulabilir. Bu sınırlı boyuttaki bir dünya, aşırı uzun listeler içermeden ilgili prosedürleri göstermek için yeterli olduğu için kullanılmıştır.

Kitabın, hareketli bir resmi ekrana getirmenin farklı aşamalarını seri bir şekilde tanıtması ve programların giderek artan güçte genel bir program oluşturmak için bir araya getirilme biçimi nedeniyle, hızda kaçınılmaz bir uzlaşma olmuştur. Bu son bölümdeki son program, önemli ölçüde daha hızlı hale gelmek için rasyonalize edilebilir ve basitleştirilebilir.

10.4. Örnek Program

10.4.1. wrld\_scn.s ve eulr\_scn.s

Burada iki ana kontrol programı vardır. Her ikisi de hareket eden nesnelerden oluşan bir manzarada serbest uçuşa izin verir ancak kullanılan görüntüleme dönüşümü türünde farklılık gösterir. Bunlardan birinde, wrld\_scn.s, hareket, gözlemcinin koordinat çerçevesinin anlık eksenleri etrafında dönüşler yoluyla joystick ve klavye aracılığıyla kontrol edilir. Diğerinde, eulr\_scn.s, joystick Euler açılarını artırır veya azaltır ve gözlemcinin referans çerçevesinin yönelimini değiştirir. Ayrıntılı kontroller şunlardır

wrld\_scn: yukarı, aşağı, sol, sağ = joystick sola yuvarlanma = fl, sağa yuvarlanma =F2

eulr scn: yukarı, aşağı, sol, sağ = joystick.

Her iki durumda da diğer fonksiyon tuşları şunlardır:

geri= F3, yavaş ileri= F4, hızlı ileri= F5, durdur= F6, iptal= F7.

10.4.2. veri\_06.s

Bu, basit 3B yapılar olan grafik ilkellerinin veri dosyasıdır. Bunlar, her bir döşemeyle ilişkili ilkelin nitelik baytının düşük nibble'ında belirtildiği data\_08.s'deki veritabanına göre manzaranın her yerine dağılmış olarak görünür. Adres ilkelindeki bir atlama tablosundan vektörlenen 6 tür (0-5) vardır. Çeşitlilik veya sayı için bir sınır yoktur; yenisini eklemek için atlama vektörlerine bir etiket daha ekleyin ve ayrıntıları listenin sonuna doldurun. İlkeldeki birincil atlama vektörleri, her belirli tür için veri tabloları olan ikincil vektörlerin bir listesine işaret eder. Belirli bir tür için veriler bir dizi liste halinde verilir:

* ikincil işaretçiler,
* içsel renkler (4 rengin 8 tonu için 0, 1, 2 veya 3),
* her çok yüzlü nesnedeki yüz sayısı,
* her yüzdeki kenar numaralarının listesi,
* tüm yüzlerdeki tepe noktası bağlantılarının listesi sırayla,
* köşelerin x, y ve z koordinatlarının üç kümesi,
* toplam tepe noktası sayısı ve
* nesnenin geçirdiği dönme türü.

Her türün gösterdiği dönme hareketinin türü, 0n değişkeninin yüksek kelimesinin en düşük baytında belirtilir (burada n tür numarasıdır) ve düşük kelime program tarafından geçerli açıyı tutmak için kullanılır ancak listede 0 olarak görünür. Dönme türü, baytta ayarlanan bit tarafından verilir:

bit 0 - nesne çerçevesinin x ekseni etrafındaki dönüş

bit 1 - dedi y

bit 2 - aynı şekilde z

böylece eş zamanlı rotasyonların herhangi bir kombinasyonu dahil edilebilir.

10.4.3. veri\_07.s

İşte, x ve y eksenleri etrafındaki pozitif ve negatif dönüşler için satır sırasına göre düzenlenmiş dört kontrol matrisi.

10.4.4. veri\_08.s

İşte 16\*16 fayans dünya birimini oluşturan 256 bayt. Programda, sarmalama gerçekleşir, böylece aşırı sol sınırın ötesindeki hareket izleyiciyi sağ sınıra döndürür. Bu anlamda, bir küre gibi, dünya "sonsuzdur". Her baytta yüksek nibble arka planın gerçek rengini verir \((0-7), aydınlatma yok) ve yüksek nibble fayans üzerinde oturan nesne türünü (0-15) verir. Programda yalnızca 6 tür kullanılır. Okuyucu kolayca yenilerini icat edebilir.

10.4.5. çekirdek\_07.s

Çekirdekteki ilk alt rutin, patch\_ext, önce gözlemcinin geçerli konumunu alır ve dünya haritası içinde yer alacak şekilde normalleştirir. Sarma işlemi burada gerçekleşir. Bunu takiben, döşeme koordinatlarındaki (Ty,Tz) konum, y ve z konumlarına 256'ya bölünerek hesaplanır. Yz düzlemi üzerinde yayılmış 16\*16 döşeme olduğunu unutmayın. Bu, gözlemcinin konumunun düzleme dikey izdüşümüdür. Daha sonra, bu konum etrafında merkezlenen 16 döşemenin nitelikleri veritabanından alınır ve her döşeme için, her birine eşlik eden 4 kelimelik kayıttaki ilk kelimenin ilk baytı olarak saklanır. Her döşemenin gözlemcinin konumundan uzaklığı, ilk kayıt kelimesinin ikinci baytında kaydedilir. Potansiyel olarak görülebilen bu döşeme koleksiyonuna yama denir.

Bunun ardından yamadaki her bir döşemede bir görünürlük testi yapılır. Buradaki test görünürlükteki bir hayal kırıklığını değil, yalnızca döşemenin merkezinin gözlemcinin önünde olup olmadığını dikkate alır. Bu test sırasında her döşeme için hesaplanan merkezi parametre, gözlemcinin önündeki uzaklığıdır (zv). Bu ayrıca daha sonra derinlik sıralaması için kayıttaki ikinci kelime olarak kaydedilir. Döşemelerin yarısından azı görünürlük testini geçer. Görünürlük sıralaması, daha sonra kayıtları gözlemciden azalan mesafe sırasına göre yerleştirmek için basitçe bir kabarcık sıralaması kullanır. Listenin en üstünde kayıtları olan döşemeler en uzakta oldukları için önce çizilecektir.

Bunu izleyen alt yordam, drwjt, görünür döşemelerin sıralı listesindeki her döşemeyi ve onun yerleşik nesnesini çizmek için verileri ayarlar ve tüm resmi çizmek için önceki tüm alt yordamları çağırır. Bu aşamada çok şey olur. Her döşemedeki arka plan, belirli bir rengin çarpı işaretidir, böylece tüm döşemeler birlikte nesnelerin oturduğu bir ızgarayı tanımlar. Arka plan her döşeme için aynı olduğundan, bir veri dosyasında saklanmak yerine doğrudan programdan girilir. Ayrıca, değişen aydınlatma olmadan sabit bir renge sahip olduğundan, zaman alıcı aydınlatma hesaplamalarını çağırmaya gerek yoktur.

Her nesnenin veri listesi veri dosyasından çekilir ve çizilmeden önce, hangi dönüş modu etkinse, dünya çerçevesindeki yeni açısı belirlenir.

10.4.6. bss\_07.s

Yeni değişkenler

10.4.7. sistem\_Ol.s

Sistemi kurmak için sadece birkaç rutin. Özellikle bakış açısı, perspektif bozulmasını azaltmak ve nesnelerin parçalarının gözlemcinin arkasına düşme olasılığını ortadan kaldırmak için zv ekseninde biraz geriye -300'e taşınır, bu sistemin çökmesine neden olmaz ancak temel çizim rutinleri geriye doğru çizimle başa çıkmaya çalışırken muhteşem bir çöp görüntüsü üretir.

Ayrıca biraz hile. Amiga bu programla genişletiliyor ve ekrandaki pencerenin (klip çerçevesi) boyutunu küçülterek işleri hızlandırmaya yardımcı oluyor, böylece resim daha küçük oluyor (oyunların neden konsol gibi görünen çok sayıda statik süslemeyle çevrili minik bir ekran gösterdiğini hiç merak ettiniz mi?).

10.4.8. çekirdek O8.s

Bu, Euler açı dönüşümünün çekirdek dosyasıdır.

10.5. Sonsöz

Ne kadar yol kat ettik? Sırada ne var? Başlangıç ​​için, bu kitapta programların seri olarak tanıtılma biçiminden kaynaklanan anormallikleri rasyonalize ederek genel program önemli ölçüde hızlandırılabilir.

Üçüncü tarafın (siz, dünya sahnesi ve uzaylı) dahil edilmesi de kalır. Şimdiye kadar grafik varlıklar, evrimlerinin nitelikleri tarafından belirlendiği anlamında statikti. Varlıklara hayat vermek, eylemlerinin programın deterministik yapısından bağımsız olarak evrimleşmesini gerektirir. Ancak bu senaryoda gerçekten yalnızca bir tane gerçekten rastgele unsur vardır - siz, gözlemci. Bu nedenle, bilgisayar içinde hayat yaratmak için varlıkların eylemlerinize yanıt vermesini sağlamak gerekir. Bu elbette tüm oyunlarda olan şeydir. Uzaylılar hedefe doğru yönelir. Üçüncü bir taraf icat etmek, gözlemcinin hareketini takip etmek için kumanda kolunun hareketlerini okumaktan daha karmaşık değildir. Üçüncü taraf durumunda kumanda kolu hareketleri yoktur, bunun yerine dünya koşullarına yanıt vardır.

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\* wrld\_scn.s

\* A multi-object scene

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\* A world scene consisting of various types of graphics primitives

\* in notion. The viewer is free to \*fly\* to any location. At any

\* position a patch consisting of 4\*4 "tiles" is visible.

\* Joystick controls Yaw and pitch. F1 and F2 controll roll

\* Don't held down keys as keyboard buffer is not cleared.

\* SECTION TEXT

opt d+

bra main

include systm\_01.s

include core\_07.s

main:

\* Initalize the system.

bsr init\_vars initialize view transform

bsr flg\_init initialize flags

loop:

\* Read input and make adjustments.

bsr swp\_scn swap the screens

bsr dircosines regenerate view matrix

bsr joy\_read see which direction to move

bsr in\_key update the speed

bsr adj\_vel adjust the velocity

\* Draw the scene

bsr scne\_drw everything to complete the picture

\* Draw the next frame

bra loop

\*SECTIOM DATA

include data\_00.s

include data\_06.s

include data\_07.s

include data\_08.s

\*SECTION BSS

include bss\_07.s

END

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Core\_07.s \*

\* subroutines for chapter 10 \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

include Core\_06.s

scne\_drw ; draw a scene of several primitives

bsr patch\_ext select the local scene

bsr sight\_tst select only the visible ones

bsr vis\_srt sort in depth order

bsr drw\_it draw them in depth order

rts

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Extract the tile patch. Put the 16 tiles in a list at patch\_lst

patch\_ext

move.w oposx,d0 observers x pos

move.w oposy,d1

move.w oposz,d2

\* Find position in world. Keep to range 4096

andi.w #$fff,d0 range x

andi.w #$fff,d1 range y

andi.w #$fff,d2 range z

move.w d0,oposx restore x etc..

move.w d1,oposy

move.w d2,oposz

move.w d1,d3

move.w d2,d4

\* Find coords of patch centre=local world origin

lsr.w #8,d1

move.w d1,Ty y coord. in 16\*16 layout

lsr.w #8,d2

move.w d2,Tz z coord

\* Coords of view frame, referenced to this origin

lsl.w #8,d1 Ty\*256

lsl.w #8,d2 Tz\*256

sub.w d1,d3 oposy-Ty\*256 = Ovy

move.w d3,Ovy

sub.w d2,d4 opoz-Tz\*256 = Ovz

move.w d4,Ovz

move.w oposx,Ovx (the height is universal)

\* Fetch the attributes of the 16 surrounding tiles from the map and calculate their world

\* coords. Store the data in a record/structure with the format:-

\* WORD 1 : HIBYTE - graphics attribute

\* LOBYTE - clear

\* WORD 2 : Voz tile centre z in view frame coords

\* WORD 3 : tile y in local world coords

\* WORD 4 : ditto z

\* Ty & Tz are the patch centre coords = local world origins.

move.w Ty,d0

move.w Tz,d1

\* A 4\*$ patch of tiles centred on the Ty,Tz are retrieved

move.w #-2,d5 z offset of start tile

lea map\_base,a0

lea patch\_lst,a1 the local list of 4\*4

move.w #3,d7 4 z values

tile\_lp1

move.w #-2,d4 reset start yoffset

move.w #3,d6 4 y values

move.w d1,d3 origin Tz

add.w d5,d3 +offset = next z

andi.w #$f,d3 stay in range 0-15

lsl.w #4,d3 \*16

tile\_lp2

move d0,d2 origin Ty

add.w d4,d2 +offset = next y

andi.w #$f,d2 stay in range 0-15

add.w d3,d2 16\*z+y = tile address in map

move.b 0(a0,d2.w),d2 fetch attribute in low byte

swap d2 of high word

clr.w d2 0 for low word

lsl.l #8,d2 everything into high word

move.l d2,(a1)+ store the first half of the record

\* Calculate the tile local coords: Ooy & Ooz. Coords are offset\*256.

movem.l d4/d5,-(sp) stack offsets

lsl #8,d4 yoffset\*256

swap d4 in high word

lsl #8,d5 zoffset\*256

move.w d5,d4 in low word

move.l d4,(a1)+

movem.l (sp)+,d4/d5 restore offsets

addq #1,d4 next y offset

dbra d6,tile\_lp2 for all tiles in this row

addi.w #1,d5 next z offset

dbra d7,tile\_lp1 for all rows

rts

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

sight\_tst

lea patch\_lst,a0 pointer to source list

lea vis\_lst,a1 list of visible tiles

lea vis\_cnt,a2 count of previous

clr.w (a2) zero count

move.w #15,d7 16 tiles in a patch

clr.w Oox all tiles are on the ground

sight\_tst1

move.w 4(a0),d0

addi.w #128,d0

move.w d0,Ooy tile

move.w 6(a0),d0

addi.w #128,d0

move.w d0,Ooz centres

movem.l d7/a0-a2,-(sp)

bsr testview is tile within filed of vision

movem.l (sp)+,d7/a0-a2

tst.b viewflag visible?

beq nxt\_tile

addq.w #1,(a2) yes, increment visible count

move.w Voz,2(a0) save the depth for sorting

move.l (a0),(a1)+ transfer 1st half to visible list

move.l 4(a0),(a1)+ 2nd half

nxt\_tile

addq #8,a0 point to next record

dbra d7,sight\_tst1 for all tiles

rts

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*Test whether the primitive is visible.

\* Tile centre (Oox, Ooy, Ooz) transformed to view coords then tested. Correct for 2^14.

testview

moveq.l #2,d6 3 rows in matrix

lea w\_vmatx,a3 init max pointer

link a6,#-6 3 words to store temporarily

move.w Oox,d3

move.w Ooy,d4

move.w Ooz,d5

sub.w Ovx,d3 Oox-Ovx rel to the view frame

sub.w Ovy,d4 Ooy-Ovy

sub.w Ovz,d5 Ooz-Ovz

tranv0

move d3,d0 restore

move d4,d1

move d5,d2

muls (a3)+,d0 \*Mi1

muls (a3)+,d1 \*Mi2

muls (a3)+,d2 \*Mi3

add.l d1,d0

add.l d2,d0 \*Mi1+\*Mi2+\*Mi3

lsl.l #2,d0

swap d0 /2^14

move.w d0,-(a6) save it

dbra d6,tranv0 repeat for 3 elements

move.w (a6)+,d3 off my stack becomes Voz

move.w (a6)+,d2 off my stack becomes Voy (centre in view frame)

move.w (a6)+,d1 off my stack becomes Vox

move.w d3,Voz

move.w d2,Voy

move.w d1,Vox

unlk a6

\* Clip Ovz. To be visible must have 50<Voz<2000

\* This test only looks at depth.

cmp.w #50,d3 test(Voz-50)

bmi notvis fail

cmp.w #2000,d3 test(Voz-2000)

bpl notvis fail

st viewflag we can see it

rts

notvis

sf viewflag can't see it

rts

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Order the visible tiles in orderof decreasing Voz (the distance of the tile centre from

\* the view frame origin). Largest Voz's (furthest) should be drawn first.

vis\_srt

move.w vis\_cnt,d7 number to do

beq srt\_quit

subq #1,d7

beq srt\_quit

subq #1,d7

\* Bubble sort

vis\_srt1

lea vis\_lst+2,a0 pointer to 1st record Voz

movea.l a0,a1

addq.l #8,a1 pointer to 2nd Voz

move d7,d6 reset count

clr.w srt\_flg

vis\_srt2

cmpm.w (a0)+,(a1)+ test(Voz2-Voz1)

ble no\_swap 1st is farther

move.l -4(a0),d0 fetch 1st record

move.l (a0),d1

move.l -4(a1),-4(a0) make

move.l (a1),(a0) 2nd the 1st

move.l d0,-4(a1) & 1st

move.l d1,(a1) 2nd

st srt\_flg

no\_swap

addq.l #6,a0 point to next record Voz

addq.l #6,a1 and the one follwing

dbra d6,vis\_srt2

tst.w srt\_flg

beq srt\_quit

bra vis\_srt1

srt\_quit

rts

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

drw\_it

\* draw the visible tiles

move.w vis\_cnt,d7

beq drw\_it\_out

subq.w #1,d7

lea vis\_lst,a0 ptr to list

drw\_it1

movem.l d7/a0,-(sp)

bsr set\_prim drw next prim

movem.l (sp)+,d7/a0

addq.l #8,a0 next record

dbra d7,drw\_it1

drw\_it\_out

rts

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Set up next primitive for drawing; pointer to record in a0.

\* 1. DO BACKGROUND

set\_prim

move.l a0,-(sp) save ptr

bsr ldup\_bkg

bsr otranw obj->world

bsr w\_tran\_v world->view

\* Background always visible at a constant illumination level

movea.l (sp)+,a0 restore ptr

move.w (a0),d0 1st word of record

move.l a0,-(sp) save pointer

lsr.w #8,d0 top byte

lsr.w #4,d0 top nibble is colour

move.w d0,col\_lst the final colours

move.w d0,col\_lst+2

bsr perspective

bsr scrn\_adj centre it

bsr polydraw

\*2. Draw the object

movea.l (sp)+,a6 restore pointer

bsr ldup\_obj

bsr otranw

bsr w\_tran\_v

bsr illuminate

bsr perspective

bsr scrn\_adj

bsr polydraw

rts

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Load background data as program data. Background is a grid.

ldup\_bkg

move.w #2,npoly 2 rectangles

move.l #$40004,snedges 4 edges in each

lea sedglst,a2 edgelist 0,1,2,3,0,4,5,6,7,4

move.l #1,(a2)+ edges 0,1

move.l #$20003,(a2)+ 2,3

move.l #4,(a2)+ 0,4

move.l #$50006,(a2)+ 5,6

move.l #$70004,(a2)+ 7,4

\* The background vertices define a cross. All x coords are zero.

lea ocoordsx,a2 vertex coords x =

move.l #0,(a2)+ 0,0

move.l #0,(a2)+ 0,0

move.l #0,(a2)+ 0,0

move.l #0,(a2) 0,0

lea ocoordsy,a2 y =

move.l #$ff800080,(a2)+ -128,128

move.l #$80ff80,(a2)+ 128,-128

move.l #$fffcfffc,(a2)+ -4,-4

move.l #$40004,(a2) 4,4

lea ocoordsz,a2

move.l #$40004,(a2)+ 4,4

move.l #$fffcfffc,(a2)+ -4,-4

move.l #$ff800080,(a2)+ -128,128

move.l #$80ff80,(a2)+ 128,-128

move.w #8,oncoords

move.w #8,vncoords

move.w #8,wncoords

\* The tile centre in the world frame is Oox=0 and the contents of the 3rd & 4th

\* words of the records.

move.w #0,Oox

move.w 4(a0),Ooy 3rd word

addi.w #128,Ooy

move.w 6(a0),Ooz 4th word

addi.w #128,Ooz

clr.w otheta no orientation

clr.w ophi

clr.w ogamma

rts

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* This has no label in the book and therefore it seems unlikely that it will ever be used.

move.w #1,npoly 1 rectangles

move.l #$4,snedges 4 edges

lea sedglst,a2 edgelist 0,1,2,3,0

move.l #1,(a2)+ edges 0,1

move.l #$20003,(a2)+ 2,3

move.l #0,(a2)+ 0,4

\* The background vertices are the corners of the tile.

lea ocoordsx,a2 vertex coords x =

move.l #0,(a2)+ 0,0

move.l #0,(a2)+ 0,0

move.l #0,(a2)+ 0,0

move.l #0,(a2)+ 0,0

lea ocoordsy,a2 y =

clr.l (a2)+

move.l #$ff00ff,(a2)

lea ocoordsz,a2

move.l #$ff,(a2)+ 0,255

move.l #$ff0000,(a2)+ 255,0

move.w #4,oncoords

move.w #4,vncoords

move.w #4,wncoords

\* The tile centre in the world frame is Oox=0 and the contents of the 3rd & 4th

\* words of the records.

move.w #0,Oox

move.w 4(a0),Ooy 3rd word

move.w 6(a0),Ooz 4th word

clr.w otheta no orientation

clr.w ophi

clr.w ogamma

rts

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ldup\_obj

\* Find out what type of object it is.

move.w (a6),d0 top word

lsr.w #8,d0 top byte

andi.w #$f,d0 low nibble is type (call it n)

lsl.w #2,d0 \*4 for offset

lea primitive,a5 ptr to vector table

movea.l 0(a5,d0.w),a5 ptr to type n lists

movea.l 4(a5),a2 pointer to npolyn

move.w (a2),d7 got it

move.w d7,npoly

subq.w #1,d7

move d7,d0

movea.l 8(a5),a0 ptr to nedge list

movea.l a0,a4 saved

lea snedges,a1 destination

move.l (a5),a2 ptr to intrinsic colours

lea srf\_col,a3 dest

obj\_lp1

move.w (a0)+,(a1)+ transfer edge numbers

move.w (a2)+,(a3)+ transfer intrinsic colours

dbra d0,obj\_lp1

\* Calculate total number of edges

move.w d7,d0 retore count

clr d1

clr d2

obj\_lp2

add.w (a4)+,d2 number of edges

addq #1,d2 and with last repeated

dbra d0,obj\_lp2

\* Move the edge list

subq #1,d2 counter

movea.l 12(a5),a0 edglstn, the source

lea sedglst,a1 dest

obj\_lp3

move.w (a0)+,(a1)+ pass it

dbra d2,obj\_lp3

\* and the coords list

movea.l 28(a5),a0 ptr to num vertices

move.w (a0),d1 num vertices

move.w d1,oncoords

move.w d1,vncoords

move.w d1,wncoords

subq #1,d1 counter

movea.l 16(a5),a0 ptr to object x

lea ocoordsx,a1

movea.l 20(a5),a2 object y

lea ocoordsy,a3

movea.l 24(a5),a4 object z

movea.l a5,a6

lea ocoordsz,a5

obj\_lp4

move.w (a0)+,(a1)+

move.w (a2)+,(a3)+

move.w (a4)+,(a5)+

dbra d1,obj\_lp4

\* Increment the rotation angle

bsr next\_rot

addi.w #128,Ooy

addi.w #128,Ooz

rts

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Increment the rotation of the object.

next\_rot

movea.l 32(a6),a0 ptr to angle and flag

move.l (a0),d0 top word is flag, bottom is angle

move.l d0,d1

andi.l #$ffff,d0 the angle

addi.w #2,d0 increment it

cmp #360,d0

blt obj\_lp5

subi #360,d0

obj\_lp5

move.w d0,2(a0) next angle

\* see what angles to rotate

swap d1

andi.w #$f,d1 flag in lo nib

\* flags are set:bit 0= xrot 1=yrot 2=zrot

lsl.w #2,d1 offset

lea rot\_vec,a0 ptr to jump table

move.l 0(a0,d1.w),a0

jmp (a0)

rot\_vec

dc.l no\_rot,rotx,roty,rotxy,rotz,rotxz,rotyz,rotxyz

no\_rot rts

rotx

move.w d0,otheta

rts

roty

move.w d0,ophi

rts

rotxy

move.w d0,otheta

move.w d0,ophi

rts

rotz

move.w d0,ogamma

rts

rotxz

move.w d0,otheta

move.w d0,ogamma

rts

rotyz

move.w d0,ophi

move.w d0,ogamma

rts

rotxyz

move.w d0,otheta

move.w d0,ophi

move.w d0,ogamma

rts

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* These are the rotations the joystick reader sends us here.

rot\_down

lea rot\_y\_neg,a0 ptr to ctrl matrix

bsr ctrl\_view

rts

rot\_up

lea rot\_y\_pos,a0 ptr to ctrl matrix

bsr ctrl\_view

rts

rot\_left

lea rot\_x\_pos,a0 ptr to ctrl matrix

bsr ctrl\_view

rts

rot\_right

lea rot\_x\_neg,a0 ptr to ctrl matrix

bsr ctrl\_view

rts

roll\_left

lea rot\_z\_neg,a0 ptr to ctrl matrix

bsr ctrl\_view

rts

roll\_right

lea rot\_z\_pos,a0 ptr to ctrl matrix

bsr ctrl\_view

rts

ctrl\_view

\* multiply the control matrix poited to by a0 by the view matrix

\* to calculate the new elements of the view base vectors.

\* 1.base vector iv

lea w\_vmatx,a1 ptr to view matrix

lea iv,a2 ptr to view frame base vector

move.w #2,d6 3 elements to iv

movea.l a1,a3 set view ptr

iv\_loop

move.w (a3),d1 next view elements

move.w 6(a3),d2

move.w 12(a3),d3

muls (a0),d1

muls 2(a0),d2

muls 4(a0),d3

add.l d2,d1

add.l d3,d1

lsl.l #2,d1

swap d1

move.w d1,(a2)+ next element in base vector

addq.l #2,a3 next column in base vector

dbra d6,iv\_loop

\*2. No need to do jv; it's calculated from the other two.

\*3. base vector kv

lea kv,a2

move.w #2,d6

movea.l a1,a3

kv\_loop

move.w (a3),d1

move.w 6(a3),d2

move.w 12(a3),d3

muls 12(a0),d1

muls 14(a0),d2

muls 16(a0),d3

add.l d2,d1

add.l d3,d1

lsl.l #2,d1

swap d1

move.w d1,(a2)+ next element in base vector

addq.l #2,a3 next column in base vector

dbra d6,kv\_loop

rts

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Set the velocity components

adj\_vel

lea kv,a0

move.w #14,d7

move.w speed,d0

lsl.w #4,d0

move d0,d1

move d0,d2

muls (a0),d0 v\*VZx

lsr.l d7,d0

add.w d0,oposx xw speed component

bpl adj1

clr.w oposx oposx must be > 0

adj1

muls 2(a0),d1 v\*VZy

lsr.l d7,d1

add.w d1,oposy yw speed component

muls 4(a0),d2 v\*VZz

lsr.l d7,d2

add.w d2,oposz zw speed component

rts

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* bss\_07.s \*

\* variables for chapter 10 \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

include bss\_06.s

\* Observer's position in world.(mod 4096)

oposx ds.w 1

oposy ds.w 1

oposz ds.w 1

\* Tile offset in 16\*16 patch

Ty ds.w 1

Tz ds.w 1

\* Tile lists

patch\_lst ds.l 32 records of 16 tiles in patch

vis\_lst ds.l 32 records of visible tiles

\* List variables

vis\_cnt ds.w 1 number of visible tiles

srt\_flg ds.w 1 set during depth sorting

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* System\_01.s \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

init\_vars:

\* set up the screens

bsr init

\* Set up the view point

move.w #100,oposx

clr.w oposy

clr.w oposz

\* and the clip frame

move.w #50,clp\_xmin

move.w #270,clp\_xmax

move.w #30,clp\_ymin

move.w #170,clp\_ymax

\* Set up view frame base vectors

\*1. iv

lea iv,a0 align view frame axes

move.w #$4000,(a0)+

move.w #0,(a0)+

move.w #0,(a0)

\*2. jv

lea jv,a0 with the world frame

clr.w (a0)+

move.w #$4000,(a0)+

clr.w (a0)

\*3.kv

lea kv,a0

move.w #0,(a0)+

clr.w (a0)+

move.w #$4000,(a0)

flg\_init:

\* Initialize flags and other variables

clr.w speed start at rest

clr.w screenflag 0=screen 1 draw, 1=screen 2 draw

clr.w viewflag

\* Move the view point to -300 on the view frame z axis

lea persmatx,a0

move.w #300,d0

move.w d0,(a0)

move.w d0,10(a0)

move.w d0,30(a0)

rts

swap\_scn:

tst.w screenflag screen 1 or screen2?

beq screen\_1 draw on screen 1, display screen2

bsr drw2\_shw1 draw on screen 2, display screen1

clr.w screenflag and set the flag for next time

bra screen\_2

screen\_1:

bsr drw1\_shw2 drar on 1, display 2

move.w #1,screenflag and set the flag for next time

screen\_2:

rts

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* data\_06.s \*

\* Data for chapter 10 \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

include data\_05.s (ensure we include data\_03.s as well).

\* The vector table of graphics primitives in 8 shades of 4 colours.

primitive:

dc.l prim0,prim1,prim2,prim3,prim4,prim5

\* Now follow the vector tables for each primitive.

prim0 ; A simple block

dc.l colrs0,npoly0,nedg0,edglst0,prm0x,prm0y,prm0z,npts0,theta0

colrs0 dc.w 1,1,1,1,1

npoly0 dc.w 5

nedg0 dc.w 4,4,4,4,4

edglst0 dc.w 0,1,2,3,0,3,2,4,5,3,5,4,6,7,5,7,6,1,0,7,1,6,4,2,1

prm0x dc.w 0,50,50,0,70,0,70,0

prm0y dc.w -6,-6,6,6,6,6,-6,-6

prm0z dc.w -6,-6,-6,-6,6,6,6,6

npts0 dc.w 8

theta0 dc.l $10000

prim1 ; An inverted pyramid

dc.l colrs1,npoly1,nedg1,edglst1,prm1x,prm1y,prm1z,npts1,theta1

colrs1 dc.w 2,2,2,2,3

npoly1 dc.w 5

nedg1 dc.w 3,3,3,3,4

edglst1 dc.w 0,1,2,0,0,2,3,0,0,3,4,0,0,4,1,0,1,4,3,2,1

prm1x dc.w 0,75,75,75,75

prm1y dc.w 0,-32,32,32,-32

prm1z dc.w 0,-32,-32,32,32

npts1 dc.w 5

theta1 dc.l $10000

prim2 ; A nugget.

dc.l colrs2,npoly2,nedg2,edglst2,prm2x,prm2y,prm2z,npts2,theta2

colrs2 dc.w 1,1,0,1,0,0,1,0,1,1,0,1,0,1

npoly2 dc.w 14

nedg2 dc.w 4,4,4,4,4,4,4,4,4,4,4,4,4,4

edglst2 dc.w 1,6,4,2,1,0,1,2,3,0,3,2,4,5,3,4,6,7,5,4,6,1,0,7,6,8,0,3,11,8,3

dc.w 5,10,11,3,5,7,9,10,5,7,0,8,9,7,8,11,13,12,8,11,10,14,13,11,10,9

dc.w 15,14,10,9,8,12,15,9,12,13,14,15,12

prm2x dc.w 40,60,60,40,60,40,60,40,20,20,20,20,0,0,0,0

prm2y dc.w -30,-10,10,30,10,30,-10,-30,-30,-30,30,30,-10,10,10,-10

prm2z dc.w -30,-10,-10,-30,10,30,10,30,-30,30,30,-30,-10,-10,10,10

npts2 dc.w 16

theta2 dc.l $70000

prim3 ; A Tee.

dc.l colrs3,npoly3,nedg3,edglst3,prm3x,prm3y,prm3z,npts3,theta3

colrs3 dc.w 2,2,2,2,2,2,2,2,2,2

npoly3 dc.w 10

nedg3 dc.w 4,4,4,4,4,4,4,4,4,4

edglst3 dc.w 0,1,2,3,0,3,2,4,7,3,4,5,6,7,4,5,1,0,6,5

dc.w 8,11,14,15,8,13,14,11,10,13,12,13,10,9,12,8,15,12,9,8

dc.w 12,15,14,13,12,10,11,8,9,10

prm3x dc.w 0,45,45,0,45,45,0,0,70,45,45,70,45,45,70,70

prm3y dc.w -10,-10,10,10,10,-10,-10,10,128,128,128,128,-128,-128,-128,-128

prm3z dc.w -10,-10,-10,-10,10,10,10,10,10,10,-10,-10,10,-10,-10,10

npts3 dc.w 16

theta3 dc.l $10000

prim4 ; A roller

dc.l colrs4,npoly4,nedg4,edglst4,prm4x,prm4y,prm4z,npts4,theta4

colrs4 dc.w 1,0,1,0,1,0,1,1

npoly4 dc.w 8

nedg4 dc.w 4,4,4,4,4,4,6,6

edglst4 dc.w 1,2,8,7,1,0,1,7,6,0,5,0,6,11,5,4,5,11,10,4,3,4,10,9,3

dc.w 2,3,9,8,2,4,3,2,1,0,5,4,6,7,8,9,10,11,6

prm4x dc.w 0,40,40,0,-40,-40,0,40,40,0,-40,-40

prm4y dc.w -8,-8,-8,-8,-8,-8,8,8,8,8,8,8

prm4z dc.w -45,-20,20,45,20,-20,-45,-20,20,45,20,-20

npts4 dc.w 12

theta4 dc.l $20000

prim5 ; Another roller

dc.l colrs5,npoly5,nedg5,edglst5,prm5x,prm5y,prm5z,npts5,theta5

colrs5 dc.w 3,2,3,2,3,2,3,3

npoly5 dc.w 8

nedg5 dc.w 4,4,4,4,4,4,6,6

edglst5 dc.w 1,2,8,7,1,0,1,7,6,0,5,0,6,11,5,4,5,11,10,4,3,4,10,9,3

dc.w 2,3,9,8,2,4,3,2,1,0,5,4,6,7,8,9,10,11,6

prm5x dc.w 0,40,40,0,-40,-40,0,40,40,0,-40,-40

prm5y dc.w -8,-8,-8,-8,-8,-8,8,8,8,8,8,8

prm5z dc.w -45,-20,20,45,20,-20,-45,-20,20,45,20,-20

npts5 dc.w 12

theta5 dc.l $40000

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* data\_07.s \*

\* Control matrices for rotation \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* +ve rotation about the view frame x axis (LEFT) by 5 degrees.

rot\_x\_pos:

dc.w 16384,0,0,0,16322,1428,0,-1428,16322

\*-ve rotation about the xv axis (RIGHT)

rot\_x\_neg:

dc.w 16384,0,0,0,16322,-1428,0,1428,16322

\*+ve rotation about the yv axis (UP)

rot\_y\_pos:

dc.w 16322,0,-1428,0,16384,0,1428,0,16322

\*-ve rotation about the yv axis (DOWN)

rot\_y\_neg:

dc.w 16322,0,1428,0,16384,0,-1428,0,16322

\*+ve rotation about the zv axis (ROLL RIGHT)

rot\_z\_pos:

dc.w 16322,1428,0,-1428,16322,0,0,0,16384

-+ve rotation about the zv axis (ROLL LEFT)

rot\_z\_pos:

dc.w 16322,-1428,0,1428,16322,0,0,0,16384

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* data\_08.s \*

\* The world map for chapter 10. Each byte gives the attribute of a size 256\*256 tile in \*

\* a 16\*16 tile world. The attributes' composition is thus:- \*

\* High Nibble : Background colour (1-7) \*

\* Low Nibble : Primitive type (0-5) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

map\_base

dc.b $62,$62,$62,$50,$41,$35,$35,$35

dc.b $35,$35,$35,$43,$45,$54,$54,$64

dc.b $62,$62,$62,$55,$42,$33,$35,$35

dc.b $35,$35,$32,$44,$45,$54,$54,$64

dc.b $52,$52,$52,$52,$44,$35,$34,$35

dc.b $35,$30,$35,$41,$44,$54,$54,$64

dc.b $45,$41,$42,$42,$42,$35,$22,$23

dc.b $23,$20,$25,$25,$44,$44,$40,$65

dc.b $33,$35,$30,$32,$32,$22,$25,$25

dc.b $25,$23,$24,$24,$35,$32,$35,$31

dc.b $35,$32,$35,$35,$32,$22,$11,$11

dc.b $10,$10,$24,$24,$33,$35,$32,$34

dc.b $20,$25,$25,$25,$20,$21,$13,$13

dc.b $13,$13,$20,$25,$25,$25,$20,$25

dc.b $24,$25,$25,$25,$21,$21,$13,$13

dc.b $13,$13,$20,$20,$25,$25,$20,$25

dc.b $20,$25,$25,$25,$22,$22,$13,$13

dc.b $13,$13,$14,$24,$25,$25,$22,$23

dc.b $25,$23,$25,$25,$23,$22,$13,$13

dc.b $13,$13,$14,$23,$25,$25,$25,$25

dc.b $31,$35,$30,$35,$31,$21,$22,$22

dc.b $20,$20,$20,$35,$35,$34,$20,$33

dc.b $45,$40,$40,$40,$41,$41,$22,$22

dc.b $22,$25,$30,$40,$40,$42,$45,$41

dc.b $40,$40,$41,$41,$44,$45,$30,$35

dc.b $35,$35,$32,$45,$40,$50,$55,$55

dc.b $61,$61,$61,$51,$53,$45,$35,$32

dc.b $35,$35,$31,$45,$40,$50,$60,$60

dc.b $61,$61,$61,$52,$55,$44,$33,$35

dc.b $33,$35,$30,$45,$40,$50,$60,$60

dc.b $61,$61,$61,$55,$51,$45,$30,$35

dc.b $32,$35,$35,$41,$45,$50,$60,$60

\* eulr\_scn.s

\* A multi-object scene

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* A world scene consisting of various types of graphics primitives

\* in motion. The viewer is free to "fly" to any location with

\* flight simulator type control from the joystick. At any

\* position a patch consisting of 4\*4 tiles is visible.

\* SECTION TEXT

opt d+

bra main

include systm\_01.s

include core\_08.s

main:

\* Initialize the system

bsr init\_vars initialize view transform

bsr flg\_init initalize flags

loop:

\* Read input and make adjustments

bsr swp\_scn swap the screens

bsr joy\_look see which directions to move

bsr angle\_update change the euler angles

bsr wtranv\_l construct the view transfers

bsr vtran\_move move it to the base vectors

bsr in\_key update the speed

bsr adj\_vel adjust the velocity

\* Draw the scene

bsr scne\_drw everything to complete the picture

\* Draw the next frame

bra loop

\* SECTION DATA

include data\_00.s

include data\_06.s

include data\_07.s

include data\_08.s

\* SECTION BSS

include bss\_07.s

END

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Core\_08.s \*

\* Subroutines for euler\_scn Chapter 12 \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

include core\_07.s previous subroutines

joy\_look:

\* Change the euler angles etheta and ephi (vtheta and vphi from chapter 10 are the same)

\* Read the joystick and update the variables accordingly

move.w $dff00c,d0 joystick 2

btst #8,d0

beq eright,dn

btst #9,d0

beq eup\_jy

bra eleft\_jy

eright\_dn:

btst #0,d0

beq eout\_jy

btst #1,d0

beq edown\_jy

bra eright\_jy

eout\_jy rts

eup\_jy:

bsr erot\_down rotate view frame down about vy axis

rts

edown\_jy:

bsr erot\_up rotate up about vy axis

rts

eleft\_jy:

bsr erot\_left rotate left aobut vx axis

rts

eright\_jy:

bsr erot\_right rotate right about wx axis

rts

erot\_down:

\* Rotate down about the yv axis. Decrement ephi (same as vphi)

move.w #-5,vph\_inc

rts

erot\_up:

\* Rotate up about the xw axis. Increment ephi (same as vphi)

move.w #5,vphi\_inc

rts

erot\_left:

\* Rotate left about the xw axis. Increment etheta

move.w #5,vtheta\_inc

rts

erot\_right:

\* Rotate right about the xw axis. Decrement etheta

move.w #-5,vtheta\_inc

rts

vtran\_move:

\* move the view transform matrix to the base vectors

\* really just a change of label

lea iv,a0

lea jv,a1

lea kv,a2

lea w\_vmatx,a3

move.w (a3)+,(a0)+ all

move.w (a3)+,(a0)+ iv

move.w (a3)+,(a0)+

move.w (a3)+,(a1)+ all

move.w (a3)+,(a1)+ jv

move.w (a3)+,(a1)+

move.w (a3)+,(a2)+ all

move.w (a3)+,(a2)+ kv

move.w (a3),(a2)

rts

Ek A: 68000 Talimat Seti

68000 talimat seti hakkında kitaplar yazılmıştır. Burada esasları özetlemekten fazlasını yapmak için yeterli alan yoktur. Motorola 16-Bit Kullanıcı Kılavuzu'nda özlü ama kapsamlı bir tartışma verilmiştir.

Assembly dili programlamanın temel özelliği, normal matematikte veya BASIC gibi üst düzey dillerde olduğu gibi soyut cebirsel değişkenlerin olmamasıdır. Şu gibi ifadeler yapmak mümkün değildir:

x=y+z olsun

her ne kadar veriler üzerinde eşdeğer manipülasyonlar yapmak mümkün olsa da.

Assembly dilinde, x, y veya z gibi isimler RAM'deki adresleri temsil eden etiketlerdir. Bu adreslerde, etiketlerle ilişkili parametrelerin geçerli değerleri olan ikili sayılar bulunabilir. Cebirsel değişkenlere bir benzerlik vardır ancak her aşamada bellekte veya işlemci kayıtlarında işlenen ikili sayının kendisidir. 68000'in adresleme modları, çeşitli talimatların yürütülmesi sırasında verilerin sistem üzerinden adreslenmesi veya yönlendirilmesi gereken tüm yollarla başa çıkmak için tasarlanmıştır.

68000 talimat seti kapsamlı ve güçlüdür. İki önemli yönü vardır: talimatların kendisi ve veri toplama ve işleme için temel çerçeveyi oluşturan adresleme modları.

A.1. Kayıtlar

68000 işlemcisi, verilere ayrılmış sekiz adet 32 ​​bit veri kaydına (D0-D7), veri ve adresler için kullanılabilen yedi adet 32 ​​bit adres kaydına (A0-A6), RAM'in son giren ilk çıkan geçici depolama alanlarına (yığınlar) işaret edecek şekilde ayarlanmış iki adet 32 ​​bit yığın işaretçisine (ikisi de A7 olarak adlandırılır ancak ayrı kullanılır, biri sistem için diğeri kullanıcı için), program ilerlemesini saymak için bir adet 32 ​​bit program sayacına ve işlemlerin sonuçlarını kaydetmek için bir adet 16 bit bayrak durum kaydına sahiptir. 32 bit kayıtlar, beş temel veri türünü işlemek için kullanılabilir: bitler, yatak rakamları, baytlar (8 bit), kelimeler (16 bit) ve uzun kelimeler (32 bit).

A.2. Adresleme Modları

Her talimat, mikrobilgisayar sisteminin bir yerindeki bir tür verinin işlenmesiyle ilgilidir: işlemcide, bellekte veya harici donanımdan. Adresleme modları, verilere erişimin birçok yolu için tasarlanmıştır. Altı temel tür vardır: Kayıt Doğrudan, Kayıt Dolaylı, Mutlak, Anında, Program Sayacı Göreli ve Kapalı, aşağıda listelenen 14 modu kapsar. Her talimat için, işlenmek üzere olan veri (bir adres olabilir) sistemin bir yerinde bulunur. Adresleme modları, bu konumun bulunma yollarını verir. En genel haliyle, bu belirlenecek adrese etkin adres (ea) denir.

A.2.1. Adresleme Modları

Anında Veri Adresleme

Veriler hemen bir sonraki kelimedir

Hızlı Veriler talimatla birlikte hemen dahil edilir

ima edilen

ea = SR, SP veya PC

Doğrudan Kayıt Olun

Adres Kaydı Doğrudan ea = An

(adlandırılmış adres sicilinde bulunan veriler)

Veri Kaydı Doğrudan ea = Dn

(adlandırılmış veri kaydında bulunan veriler)

Mutlak Veri Adresleme.

Mutlak Kısa ea = (sonraki kelime)

(veri bir sonraki kelimede verilen adrestedir ve talimatı takip eder)

Mutlak Uzun ea = (sonraki 2 kelime)

Dolaylı Adreslemeyi Kaydet

Kayıt Dolaylı ea = (An)

(veri belirtilen adreste belirtilen adrestedir) Artış Sonrası Kayıt Dolaylı ea = (An)+

((An) olarak, sonra kayıt artışını yap) Kayıt Azaltma Dolaylı ea = -(An)

(An gibi ama ön azaltma kaydı)

ea = d16(An) Ofsetiyle Dolaylı Kayıt

((An) artı bir kelime uzunluğu eklemesi olarak)

Ofset ea = d8(An,Xn) ile Dolaylı Endeksli Kayıt

(As (An) artı bir bayt uzunluğu eklemesi ve bir dizin görevi gören adres veya veri kaydının içeriğiyle birlikte)

Kayıt dolaylı adreslemenin önemli bir versiyonu, d16(An) ve d8(An,Xn)'de An yerine program sayacının kullanıldığı PC bağıldır. Bu, geçerli program sayacına göre bellek konumlarına referans sağlar ve konumdan bağımsız kod üretmek için kullanılır. Derleyici aynı sonuca ulaşan yeniden konumlandırılabilir kod ürettiğinden bu kitapta kullanılmaz.

A.3. Talimat Seti

Genel olarak talimatlar kendilerine bir kaynak işlenen ve bir hedef işlenen ile ilişkilendirilmiştir. Bunların gerçekte ne anlama geldiği, özellikle talimata bağlıdır, örneğin bir MOVE talimatında tam olarak ima ettikleri şeyi yaparlar - kaynak ve hedef etkin adreslerini sağlarlar. Bir ADD talimatında eklenecek iki sayının adreslerini verirler. Bu işlenenler talimatı aynı satırda takip eder. Talimatın kendisi cümlenin fiili gibidir.

Ek olarak, talimatın nitelikleri vardır. Bunlar, talimata bağlı olarak bayt, sözcük veya uzun sözcük türlerinden biri veya daha fazlası olabilen izin verilen veri boyutlarıdır. Ayrıca, talimatın bir sonucu olarak, koşul kodu (durum) kaydında belirli bayraklar ayarlanacak veya temizlenecektir.

Aşağıdaki liste, ana talimat tipleri için derleyici anımsatıcılarını verir.

| **Hafıza Eylemi** | **Hafıza Eylemi** |
| --- | --- |
| ABCD, uzatma ile ondalık sayıyı topla | EKLE ekle |
| VE mantıksal ve | ASL aritmetik sola kaydırma |
| ASR aritmetik sağa kaydırma | Bcc dalı şartlı olarak |
| BCHG bit testi ve değişimi | BCLR bit testi ve temizleme |
| BRA şubesi her zaman | BSET bit testi ve ayarı |
| BSR dalı alt yordama | BTST bit testi |
| CHK kayıt defterini sınırlara göre kontrol edin | CLR temiz işlenen |
| CMP ortaya çıkıyor | DBcc test koşulu, azaltma ve dallanma\* |
| DIVS imzalı bölme | DIVU imzasız bölme |
| EOR özel VEYA | EXG değişim kayıtları |
| EXT işareti uzatır | JMP atlamak |
| JSR alt yordama atlıyor | LEA etkili yük adresi |
| BAĞLANTI bağlantı yığını | LSL mantıksal sola kaydırma |
| LSR mantıksal sağa kaydırma | HAREKET et hareket et |
| MOVEM birden fazla kaydı taşı | MOVEP çevresel verileri taşı |
| MULS imzalı çarpım | MULU imzasız çarpma |
| NBCD ondalık sayıyı uzatma ile olumsuzlama | NEG reddedildi |
| NOP işlem yok | Birinin tamamlayıcısı değil |
| VEYA mantıksal veya | PEA etkili bir adrese baskı yapıyor |
| RESET harici aygıtları sıfırla | ROL uzatarak sola döndür |
| ROR uzatma ile sağa döner | ROXL uzatarak sola döndür |
| ROXR sağa doğru uzatarak döndür | RTE istisnadan geri döndü |
| RTR geri dön ve geri yükle | Alt rutinden RTS dönüşü |
| SBCD ondalık sayıyı uzatma ile çıkarma | Scc Koşullu ayarlamayı görün\* |
| DUR dur | Alt çıkarma |
| SWAP takas veri kaydı. | TAS testi ve operand ayarlama |
| Tuzak tuzak | Taşmada TRAPV tuzağı |
| TST testi | UNLK bağlantısını kaldır |

* Durum kodlarının listesi aşağıda gösterilmektedir:

A.3.1. Durum Kodları

|  |
| --- |
| cc taşıma temizle |
| CS taşıma seti |
| EQ eşit |
| F yanlış (asla doğru değil) |
| GE büyük veya eşit |
| GT'den büyük |
| Merhaba yüksek |
| LE daha az veya eşit |
| LS düşük veya aynı |
| LT'den az |
| Ml eksi |
| HİÇBİRİ EŞİT DEĞİL |
| PL artı |
| T her zaman doğrudur |
| VC taşması yok |
| VS taşması |

Koşul kodları DBcc ve Bcc gibi talimatları takip eder, ancak dikkatli olun! Kodlar bir hesaplamanın sonucunu (hedef işlenen) - (kaynak işlenen) sırasına göre test eder ve sonucu (varsa) (hedefe) yerleştirir.

Döngü işleme için kullanılan DBcc, koşul doğruysa bir sonraki talimata gidecektir, düz dallanma için kullanılan Bcc ise koşul doğruysa dallanacaktır (ve yanlışsa bir sonraki talimata gidecektir).

En belirgin döngü talimatı DBRA (bir sayacı azalt ve -1 olana kadar dallan) aslında 68000 setinde yoktur. Ancak bunun yerine DBF (azalt ve dallan, asla doğru değil) aynı sonucu elde eder. Çoğu derleyici, insanlığa bir hizmet olarak DBRA'yı yine de uygular (ancak derlemede DBF'ye dönüştürür).

A.4. Talimat Türlerinin Çeşitleri

İşte ana tiplerin ek varyasyonları. En önemlileri, daha hızlı "Quick" ve "Immediate" versiyonlarına atıfta bulunan -Q ve -I son ekleridir; Quick, ikisinin daha hızlısıdır.

|  |  |
| --- | --- |
| EKLENDİ | adres ekle |
| GÜLE GÜLE | hemen ekle |
| EKQ | hızlı ekle |
| EKLENTİ | genişleterek ekle |
| ANDY | acil |
| ANDI'den CCR'ye | hemen koşul koduna |
| ANDI'den SR'ye | hemen durum reg'e |
| CMPA | adresi karşılaştır |
| CMPI | hemen görünür |
| EORİ | özel VEYA hemen |
| EORI'den CCR'ye | özel VEYA anında koşul kodlarına |
| EORI'den SR'ye | özel VEYA anında durum kaydına |
| HAREKET | Adresi taşı |
| HAREKET | hızlı hareket et |
| CCR'ye TAŞIN | durum kodlarına geç |
| SR'ye TAŞIN | durum kaydına geç |
| SR'den TAŞIN | durum kaydından hareket et |
| USP'ye GEÇİN | kullanıcı yığın işaretçisine git |
| NEGX | kullanıcı yığın işaretçisine git |
| VEYA VEYA | acil |
| ORI'den CCR'ye VEYA | hemen koşul kodlarına |
| VEYA SR VEYA | anında durum kaydı |
| ALTIN ​​ALT | Adresi çıkar |
| SUBQ ALTIN | hızlı çıkar |
| ALTIN ​​KONUSUNDA | acil alt simge |
| SUBX ALTIN | uzatarak çıkar |

Ek B: Devpac Montajcısı

Birçok iyi derleyici mevcuttur. Bu kitaptaki programları geliştirmek için Hisoft'un Devpac Amiga 2 Assembler/Debugger'ı kullanılmıştır. Bu ekte yer alan şey, özellikle yararlı olduğu bulunan küçük bir alt kümedir.

Bir programın düzenlenmesi, birleştirilmesi, çalıştırılması ve hata ayıklamasının tek bir ortamda yapılmasını sağlar. Bu, programların en hızlı şekilde geliştirilmesini sağlar.

B.1. GenAm2

Bu, birleştirilmiş editör, derleyici ve hata ayıklayıcıdır. GenAm2 içinde programlar yazabilir, çalıştırabilir ve hata ayıklayabilirsiniz.

B.1.1. Editör

Bu, tüm programda serbestçe dolaşmanıza izin veren kullanıcı dostu bir ekran düzenleyicisidir. Sekmeler, boşluklarla ayrılmış aşağıdaki alanlardan oluşacak olan talimat satırında uygun sütun konumlarına ayarlanabilir:

etiket mnemonik işlenen(ler) yorum

Etiket aslında RAM'deki bir adrestir, ancak programda kullanıcı dostu bir kelime olarak görünür ve genellikle programla ilgili bir anlamı vardır. Örneğin, programın tekrarlayan bir döngüde döndüğü noktaysa, basitçe "döngü" olabilir. Talimat anımsatıcıları ve işlenenler Ek 1'de ele alınmıştır. Yorum alanı, programın ilerleyişinin kolayca anlaşılabilmesi için neler olup bittiğini bilgilendirici bir şekilde açıklamalıdır. Bir örnek şu olabilir:

loop move.w d0,(a0) save the flag

B.1.2. Dosyada Hareket Etme

AmigaBir dosya hakkında kaba hareketler , tuşu (sağ taraftaki A taslağı) kullanarak kolayca yapılabilir . Bir dosyanın başına (üst) veya sonuna (alt) gitmek için sırasıyla Amiga+T veya Amiga+ tuşlarına basın.B

Ekran içindeki hareketleri kontrol etmek için imleç tuşları kullanılabilir.

B.1.3. Metin Düzenleme

ControlTüm satırlar +Y tuşuna basılarak silinebilir ve Control+U tuşuna basılarak geri yüklenebilir (tekrarlayan satırlar için kullanışlıdır). Bir satır içinde silme Backspace (geri) veya Delete (ileri) tuşuna basılarak yapılabilir.

B.1.4. Metin Hareketi

En kullanışlı olanaklar arasında metin bloklarını işleyenler yer alır. Önce imleci bloğun başına getirin ve tuşuna basın F1. Bloğun sonuna gidin ve tuşuna basın F2. İşaretli bir blok çeşitli şekillerde işlenebilir (Help bunları listeler):

F3bir bloğu kaydeder; F4onu (imlecin olduğu yere) kopyalar,

Shift+F4 bunu blok tamponuna kaydeder ve buradan bir sonraki dosyaya yapıştırılabilir,

Shift+F3 veya Shift+F5 onu siler (ama aynı zamanda bir hata yapmanız durumunda onu blok tamponuna kaydeder!),

F5bloğa (imlecin olduğu yere) yapıştırır.

Amiga+W yazdır.

B.1.5. Montaj

Bir program birkaç şekilde birleştirilebilir. Sadece bir araya getirilip getirilmeyeceğini görmek için Output to None seçeneğini seçin. Bu, ilk denemede denenebilecek en iyi şeydir. Bir programı çalıştırmak ve hata ayıklamak için Output to Memory seçeneğini seçin. Bir araya getirilen programı bağımsız olarak çalıştırmak için Output to Disk seçeneğini seçin ve dosya uzantısı olarak .PRG kullanın. Bu kitaptaki 6. Bölümden sonraki programlar için, yer kalmaması için bunları diske bir araya getirmek muhtemelen en iyisidir. Örneğin, daha sonra CLI'dan yürütülebilir programlar olarak çalıştırılırlar.

B.1.6. Seçenekler

Montajın nasıl gerçekleşeceğini etkileyen birçok seçenek mevcuttur. OPT-D seçeneği (kaynak dosyanın en üstüne yazılır ancak gerçek programa bir BRA'dan sonra gelir) çok kullanışlıdır ve etiketleri hata ayıklayıcıda tutar, bu da programı takip etmeye muazzam derecede yardımcı olur.

B.1.7. Yönergeler

Derleyici yönergesi mnemotekniklerine benzer bir görünüme sahip olan ancak derleyiciye özgü olan derleyici yönergeleri oldukça standarttır. Etiketlerin değerlerini sabitlemek, sabitlerin (tablolarını) kurmak ve değişken alanını kurmak için kullanılan EQU (veya =), DC, DS gibi yaygın olanlar örnek programlar boyunca yaygın olarak kullanılır. Ayrıca derlemede dosyaları çekmek için yaygın olarak kullanılan INCLUDE yönergesidir. Bu, kitabı ve genel programı aşamalar halinde oluşturmayı mümkün kılmıştır. Yönergelerin nasıl kullanıldığını programların kendileri en iyi şekilde gösterir.

B.1.8. Hata ayıklama

Tüm assembly dili programlarında hatalar vardır. Genellikle programları yazmaktan çok hata ayıklamaya daha fazla zaman harcanır ve bu nedenle iyi bir hata ayıklayıcıya sahip olmak faydalıdır.

Hata ayıklayıcı aslında MonAm olarak adlandırılır ve bağımsız bir program olarak veya Editör içinde kullanılabilir. Editör içinde kullanmak düzenleme, birleştirme, çalıştırma, hata ayıklama döngüsünü tamamlar. Büyük olasılıkla bir programda tek adımla ilerlemek ve 68000 kayıtta ve bellekte neler olduğunu izlemek isteyeceksiniz. Üç pencere kayıt içeriklerini, program sayacının geçerli adresi etrafındaki sökülmüş bir program bölümünü ve belleğin seçili bir bölümünün içeriklerini görüntüler. Dördüncü küçük pencere mesajları iletir. Adresleri ve kayıt içeriklerini değiştirme amacıyla, herhangi bir görüntüleme penceresi Tab'i açıp kapatarak etkinleştirilebilir.

B.1.9. Programları Çalıştırma

Bir programı izlemenin birçok yolu vardır. İşte bunlardan bazıları:

Ctrl+Z veya [Ctrl + Y] tek adım; her talimat yürütülür  
Ctrl+T tek adım; BSR'leri, JSR'leri, LineA'yı, Traps'i atlar  
Ctrl+A tek adım; bir sonraki talimattan sonra bir kesme noktası yerleştirir  
(DBF'leri (DBRA'ları) atlamak için yararlıdır)  
Runçalıştırma türü için bir istem üretir, örneğin  
Gbir sonraki kesme noktasına tam hızda çalıştırma

B.1.10. Kesme Noktaları

Bunlar programı belirli adreslerde durdurmanıza olanak tanır. Programın akışını farklı çalışma modlarında kontrol ederler. İşte basit kontroller:

Amiga+B bir adreste kesme noktası ayarla  
Ctrl+K ayarlanan tüm kesme noktalarını temizle Yardım ve kesme noktalarını göstermek  
Uiçin çalıştırılacak bir adres ister  
Help

B.1.11. Çeşitli

Control+C MonST  
Lliste etiketlerini sonlandır  
Pyazdır (etkin pencere)  
Madresi değiştir  
Amiga+A başlangıç ​​adresini ayarla (etkin pencere)  
Amiga+R adlandırılmış kaydın içeriğini değiştir

B.1.12. Böcek Avı

Bu deneyimle öğrenilen bir beceridir. En faydalı ipucu, programları iyice denemeden önce kontrol etmektir. Programları yapılandırılmış bir şekilde, modüller halinde oluşturmaya çalışın, böylece hepsi bir araya getirilmeden önce her biri bağımsız olarak iyice test edilebilir. Hataları bulmak için Hata Ayıklayıcı'ya güvenmeyin. O zamana kadar programın her bir parçasının ne için olduğunu unutmuş olacaksınız. Acele etmeyin; bir saat "hata ayıklama" ve on saat hata ayıklama harcamayın!

En yaygın hatalardan biri veri yolu hatasıdır. Bu, program sayacının kendisini belleğin yanlış bir bölümüne işaret ederken bulmasıdır. Bu genellikle Yığın'ın düzeninin bozulmasından kaynaklanır, özellikle de bir alt rutinden bir dönüş adresi gerektiğinde. Alt rutin sırasında Yığını nasıl kullandığınızı görmek için bakın.

Ek C: Sayı Sistemleri

C.1. İkili

Bilgisayarlar, ya kapalı (0) ya da açık (1) olan elektronik anahtarlardan yapılır. Bu birimlerden oluşturulabilen sayı sistemine ikili (taban 2) denir, yani 2'den; 10'un kuvvetlerine giden sisteme ise onluk (taban 10) denir. İkili sistemde sayılar 2'nin kuvvetlerinden oluşturulur. Örneğin:

13 10 = 1\*2 3 + 1\*2 2 + 0\*2 1 + 1\*2 0

Sayıları bu uzun biçimde yazmak yerine, genellikle sadece 2'nin kuvvetlerinin katsayılarını sütunlara yerleştirmek tercih edilir. Sağdan etiketlenen sütun numarası, 2'nin kuvvetini verir. Bu nedenle 11 sayısı şu şekilde yazılır:

13 10 = 1011 2

İkili sayıdaki birimlerden her biri ikili basamak veya kısaca bit olarak adlandırılır. Dört bitlik gruba "nibble" denir ve özellikle bunu sık kullanan assembly dili programcıları tarafından çok sevilir.

8 bitlik bir grubun da özel bir adı vardır, "bayt", yaygın kullanımı büyük ölçüde verileri bayt olarak aktaran 8 bitlik mikro bilgisayarların çağına dayanır. 68000 gibi daha yeni 16 bitlik mikroişlemcilerde (bu mikroişlemci etiketleme şeması veri yolunun boyutuna atıfta bulunur), 16 ve 32 bitlik gruplar yaygın olarak kullanılır, bunlara sırasıyla "kelimeler" ve "uzun kelimeler" denir.

C.2. Hexadecimal (kısaca hex)

İnsanlar 10'un kuvvetleriyle sayarlar (muhtemelen 10 parmakları olduğu için) ve 2'nin kuvvetleriyle saymayı doğal bulmazlar. Ancak, özellikle bellek konumları incelendiğinde, assembly dili programcıları için ikili sistemle bir bağlantı gereklidir. Bu amaçla genellikle onaltılık sayı sistemi kullanılır. Bu sistemde, nibble'lar tek sembollere kısaltılır. 9'a kadar olan değerler için sıradan onluk sayılar kullanılır, ancak 10 ila 15 (bir nibble'ın maksimum değeri) değerleri için yeni sembollere ihtiyaç vardır. Burada büyük bir fırsat kaçırılmıştır. Yeni bilgisayar çağı sembolleri icat etmek yerine, alfabenin A, B, C, D, E, F harfleri kaçırılmıştır. Onaltılık, 16 tabanında anlamına gelir.

Üç sistemde sırasıyla ikili, onlu ve onaltılı sistemde eşdeğerlik şudur:

| **İkili** | **Dinar** | **onaltılık** |
| --- | --- | --- |
| 0000 | 0 | 0 |
| 0001 | 1 | 1 |
| 0010 | 2 | 2 |
| 0011 | 3 | 3 |
| 0100 | 4 | 4 |
| 0101 | 5 | 5 |
| 0110 | 6 | 6 |
| 0111 | 7 | 7 |
| 1000 | 8 | 8 |
| 1001 | 9 | 9 |
| 1010 | 10 | İLE |
| 1011 | 11 | B |
| 1100 | 12 | C |
| 1101 | 13 | D |
| 1110 | 14 | VE |
| 1111 | 15 | F |

C.3. Negatif Sayılar

İkili sistemde negatif sayıları kavramak zordur. Bunun nedeni, eksi işareti için ayrılmış özel bir sembol olmaması ve sayının kendisinde kodlanması gerektiğidir. Aşağıdaki şekilde yapılır.

Basitleştirmek için, sadece nibble büyüklüğündeki sayılarla çalıştığımızı varsayalım (aslında 68000'de sadece bu boyuttaki sayıları işlemek için herhangi bir talimat yoktur, bir nibble daha büyük bir sayının parçası olmalıdır). Negatif sayılarla işlem yapmak için toplam olası aralık, 0-15, eşit şekilde bölünür. 0-7 dahil (8 sayı) aralığı pozitifler için ve 15-8 dahil (yine 8 sayı) aralığı negatifler için (-1 ila -8 aralığı) ayrılmıştır. Kulağa geldiği kadar aptalca değil. Negatif bir sayı 0'dan geriye doğru sayarak elde edilir. 0'ın altında hiçbir şey yoksa yapılacak en iyi şey en üste gidip geri saymaktır. Pratik anlamda bu iyi bir yöntemdir çünkü tüm negatif sayıların en üst biti ayarlanmıştır. En üst bit dikey hale getirilmiş bir eksi işareti gibidir. Bu kural için süslü bir isim var: 2'nin tamamlayıcısı Bir sayının negatifini almak için basit bir tarif var: Sayıyı ikili sisteme yaz, tüm l'leri 0'lara ve 0'ları l'lere çevir ve sonra 1 ekle. Deneyelim. -2'nin aslında 14 olduğunu biliyoruz, işte kontrol:

Adım 1

+2 0010'dur

Adım 2 (2'nin tamamlayıcısı)

bitleri değiştir 1101

ve 1110'u elde etmek için 1 ekleyin

bu da 14'tür ve dolayısıyla doğrudur.

Negatif sayıları etiketlemenin 2'nin tamamlayıcısı yöntemi her boyutta işe yarar: bayt, sözcük ve uzun sözcük. Ancak uyaralım, sayının -2 olduğunu ve 14 olmadığını yalnızca siz biliyorsunuz, bilgisayar bilmiyor! Neler olup bittiğini takip etmenize yardımcı olmak için 68000, en üst biti bir işaret biti olarak ele alan imzalı talimatlar adı verilen talimatlara sahiptir. Sayıları yalnızca pozitif olarak ele alan başka, işaretsiz talimatlar da vardır. Bunlar yardımcı olur, ancak programcının sayıların ayrılan aralığı aşmamasını ve genellikle patolojik sonuçlarla işaret değiştirmemesini izlemesi gereken birçok durum vardır.

Montaj dilinde farklı sayı tipleri farklı önekleriyle ayırt edilir:

onlu - yok; ikili - %; onaltılık - $ .

Ek D: Yonga Kayıtları

Aşağıda özel veya özel donanım "Agnus", "Denise" ve "Paula" tarafından kullanılan kayıtların adreslerinin kısa bir listesi bulunmaktadır. Adresler, taban adresi SdffOOO'dan ofsetler olarak verilmiştir. Genel olarak çip kayıtları ya salt okunur ® ya da salt yazılır (W) ya da bazı durumlarda strobe'dur (S) (yazma ile tetiklenir) ve yanlış olanı yapmaya çalışmak sorun yaratacaktır. Aşağıda sadece bu kitaptaki programlarda görünen kayıtların birkaçı listelenmiştir. Daha fazla ayrıntı için Amiga Donanım Referans Kılavuzuna bakın. Bir diğer yararlı kitap ise Abacus tarafından yazılan "Amiga Sistem Programcısının Kılavuzu"dur, bir Data Becker Kitabı.

| **KAYIT OLMAK** | **ADRES** | **Okunabilir/Yazılabilir** | **İŞLEV** |
| --- | --- | --- | --- |
| BLTAFWM | 44 dolar | B | Blitter ilk kelime maskesi kaynak A |
| BLTALWM | 46$ | B | Kaynak A için son söz maskesi Blitter |
| BLTKONO | 40 dolar | B | Blitter kontrol kayıt numarası 0 |
| BLTCON1 | 42 dolar | B | Blitter kontrol kayıt numarası 1 |
| BLTSIZE | 58 dolar | B | Blok boyutu blit'e |
| BLTCMOD | 60 dolar | B | Blitter kaynak C modülü |
| BLTBMOD | 62 dolar | B | Blitter kaynak B modülü |
| BLTAMOD | 64 dolar | B | Blitter kaynak A modülü |
| BLTDMOD | 66 dolar | B | Blitter hedef D modülü |
| BLTCPTH | 48 dolar | B | Blitter kaynağı Bir işaretçi |
| BLTBPTH | 4C$ | B | Blitter kaynağı B işaretçisi |
| BLTAPTH | 50 dolar | B | Blitter kaynağı Bir işaretçi |
| BLTDPTH | 54 dolar | B | Blitter hedefi D işaretçisi |
| BPL1 MOD | 108 dolar | B | Tek düzlemler için bit düzlemi modülü |
| BPL2MOD | 10A$ | B | Eşit düzlemler için bit düzlem modülü |
| BPLKONO | 100 dolar | w | Bit düzlemi kontrol kaydı 0 |
| BPLCON1 | 102 dolar | w | Bit düzlemi kontrol kaydı 1 |
| BPLCON2 | 104 dolar | w | Bit düzlemi kontrol kaydı 2 |
| BPL1PTH | $0E0 | w | Bit düzlemi işaretçilerinin başlangıcı |
| Ben renklendiriyorum | 180 dolar | w | Renk tablosunun başlangıcı |
| polis1lc | 80 dolar | w | Copperlist 1 adresi |
| polis2lc | 84 dolar | w | Copperlist 2 adresi |
| polisjmp1 | 88 dolar | S | Liste 1 yeniden başlatma strobe |
| polisjmp2 | 8a dolar | S | Liste 2 strobe'u yeniden başlat |
| diwstrt | 8e dolar | w | Başlatma penceresi gösterimi |
| diwstop | 90 dolar | w | Vitrin durdurucu |
| ddfstrt | 92 dolar | w | Veri alma başlangıcını görüntüle |
| ddfdurdur | 94 dolar | w | Veri alma işlemini durdur |
| DMACON | 96$ | B | DMA durumunu ayarla |
| VPOSR | $4 | R | Dikey ışın konumunu oku |

Ek E: Vektörler ve Matrisler

Vektörler ve matrisler birlikte gider. Vektörler için seçilen kural, matrisler için de kuralı belirler.

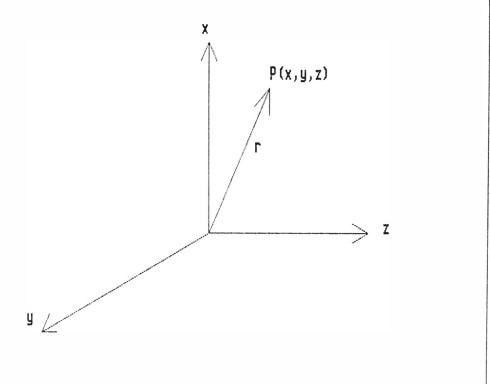
E.1. Vektörler

Bir vektör, uzayda bir konumu belirtmenin özlü bir yoludur. Konum, orijin adı verilen sabit bir konumdan ölçülür. Uzay 3 boyutlu olduğundan konum, orijinden belirli mesafeler ileri, yanlara doğru sağa ve yukarıya doğru hareket ettirilerek belirlenir (sırasıyla negatif mesafeler geriye, sola ve aşağıya karşılık gelir). Matematik dilinde bu, Kartezyen koordinat sistemindeki tüm yer değiştirmeleri ölçmek anlamına gelir. Uzaydaki bir konum daha sonra, ona ulaşmak için kat edilmesi gereken dik açılardaki üç eksen boyunca mesafelerle belirlenir. Vektör gösterimi, bu bilginin sunulma biçiminden kaynaklanır. Üç eksen boyunca P noktasına olan yer değiştirmeler sırasıyla x, y ve z ise, Şekil A6.1'de gösterildiği gibi orijinden P'ye uzanan r vektörü, vektör gösteriminde şu şekilde ifade edilebilir:

r = xi + yj + zk

Vektörleri (hem büyüklük (büyüklük) hem de yön) olanlarını, yalnızca boyuta sahip olan sıradan sayılardan ayırmak için kalın yazmak yaygındır. Burada birim veya taban vektörleri olarak adlandırılan i, j ve k, x, y ve z eksenlerini işaret eden işaret levhalarıdır ve xi terimi "x ekseninin yönünde x mesafesine git" anlamına gelir, vb. Bunlar kendi başlarına boyut (büyüklük) birliğe eşit vektörlerdir.

i, j ve k aslında yalnızca yer değiştirmenin üç bileşenini ayırt etmeye yaradığından, sıra korunduğu sürece bunları şemadan çıkarabiliriz. Üç bileşen, sütun vektör gösterimindeki matrislerle çarpmaya hazır parantezlerin içinde sırayla dahil edilebilir:



*Şekil A5.1 Kartezyen koordinatlarda bir vektör*

X

r = y

z

Vektörleri temsil etmenin tek yolu bu değildir. Bilgisayar grafiklerinde bunları satır gösteriminde temsil etmek yaygındır

r = (xyz)

Kullanılan kural, matrislerin yazılma biçimini belirler. Bu kitapta sütun vektörleri kullanılır çünkü bu, bilim ve mühendislikte daha yaygındır ve bu nedenle genel okuyucuya daha aşina olma olasılığı yüksektir. Kurallar arasında geçiş yapmak yorucudur ancak oldukça zahmetsizdir.

E.2. Matrisler

Bilgisayar grafiklerinde sıklıkla görülen dönme dönüşümleri sonucunda nesnelerin koordinatları belirli bir şekilde değişir. Bir nokta P(x,y,z), Şekil A6.2'de gösterildiği gibi bir eksen etrafında dönme sonucunda yeni bir konuma P'(x',y',z') hareket edecektir. Yeni bileşenlerin her biri, bir dizi doğrusal denklemdeki tüm eski bileşenlerle ilişkilidir:

x' = M11.x + M12.y + M13.z

y' = M21.x + M22.y + M23.z

z' = M31.x + M32.y + M33.z

Burada M'ler orijinal bileşenlerin oranlarını veren sayılardır ve bir matris M'nin elemanlarıdır. Önemli olan, matris elemanlarının dönüşe benzersiz bir şekilde ilişkili olmasıdır, böylece aynı eksen etrafında aynı şekilde döndürülen herhangi bir diğer noktanın yeni bileşenleri aynı matris M tarafından belirlenir. Matris ve vektörlerin çarpım kurallarını kullanarak, M'nin elemanlarını vektörün x, y ve z bileşenlerinden ayırarak bunu vurgulayabiliriz. Ürün şu şekilde yazılır:

X' = ​​​​M11 M12 M13 X

r' = M21 M22 M23 y

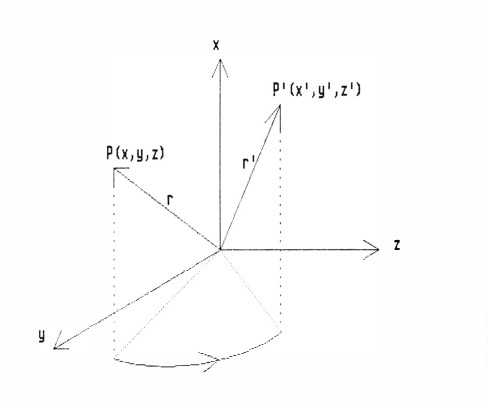
z' = M31 M32 M33 z

Bu şekilde yazılan matris çarpımı, yeni koordinatları hesaplamaya başladığımızda gerçekten önemli olan doğrusal denklemler kümesi için sadece kısaltma notasyonudur. Ancak bu şekilde yazılması, hesaplandıktan sonra matris M'nin herhangi bir noktayı aynı şekilde döndürmek için kullanılabileceğini açıkça ortaya koyar. Daha da özlü bir kısaltmayla dönüşümü şu şekilde özetleyebiliriz:

r' = Bay

Burada çarpım matris çarpımıdır ve sıradan bir sayı çarpımı değildir.

Bu kısa çarpımı denklemler kümesine geri dönüştürmek için vektörün üç satırı ve bir sütunu, matrisin ise üç satırı ve üç sütunu olduğunu gözlemleyin. Dönüştürülmüş vektör r'nin en üst satırını (x') oluşturmak için, sırayla M'nin en üst satırındaki her bir elemanı r vektörünün her bir satırıyla çarpın ve bunları toplayın. r'nin ikinci satırı, M'nin ikinci satırındaki her bir elemanın r satırlarıyla çarpımından hesaplanır ve bu şekilde devam eder (vektörlerin satır gösteriminde çalışıyor olsaydık her şey tam tersi olurdu). Matris çarpımının bu anlamı öğrenilmesi gereken bir şeydir.



*Şekil A5.2 P noktası P' noktasına aktarıldı*

E.3. Vektör Ürünleri

E.3.1. Skaler (Nokta) Ürün

Vektörler aslında sadece bir kısaltma ve geometri yapmanın oldukça düşündürücü bir yoludur. Kartezyen bir sistemdeki bir P(x,y,z) noktası, orijinden P noktasına kadar uzanan bir r vektörüyle temsil edildiğinde çok daha önemli görünür. Başka bir P' (x' ,y' ,z') noktası da benzer şekilde P' vektörüyle temsil edilir.

Çoğu zaman bu iki vektör arasındaki açıyı, θ, bilmek isteriz (önceki bölüme geri dönersek, bu vektör P'nin dönüş açısı olabilir). Bulunması en basit olanın θ'nin kosinüsü olduğu ortaya çıkar ki bu da

cosθ = (xx' + yy' + zz') / √((x^2 + y^2 + z^2).(x'^2 + y'^2 + z'^2))

Paydadaki faktörler karmaşık görünür ancak bunlar sadece Pisagor teoreminin 3 boyutlu bir versiyonu kullanılarak hesaplanan iki vektörün büyüklükleridir. Payda, iki vektörün bileşenlerinin çarpımlarının toplamıdır. Böyle bir çarpım geometride sıklıkla meydana geldiğinden özel bir sembol ve isim verilmiştir. Skaler veya nokta çarpımı olarak adlandırılır ve şu şekilde yazılır:

rr' = xx' + yy' + zz'

Skaler çarpım olarak adlandırılır çünkü iki vektörden skaler bir cevap üretir. Bir vektörün büyüklüğünü her zaman bir kareler toplamının karekökü olarak yazmak yerine, ki bu yorucudur, onu vektörle aynı sembolle ama kalın yazı olmadan göstermek yaygındır. Dolayısıyla şey şu şekilde verilir:

cosθ = (rr')/rr'

burada r = |r| = √(x^2 + y^2 + z^2) ve aynı şey r' için de geçerlidir.

|r| işlemi 'r'nin büyüklüğü' anlamına gelir.

Skaler çarpım rr'nin cosθ ile orantılı olduğunu ve en önemlisi cosθ ile aynı işarete sahip olduğunu fark edin. Kosinüsün işareti, iki vektörün paralel (aynı yönü işaret eden) veya antiparalel (zıt yönleri işaret eden) olup olmadığını test etmek için çok yararlı bir testtir ve yüzeylerin görünürlüğünü test etmede önemli bir rol oynar.

E.3.2. Vektör (Çapraz) Ürün

Bu, yeni bir vektör üreten iki vektörün çarpımıdır. Bir kez daha yararlı bir uygulamaya dayanmaktadır. Bu durumda, her iki orijinal vektöre dik (dik açılarda) olan vektörü üretir. Bunu ifade etmenin başka bir yolu, yeni vektörün iki ürün vektörünü içeren düzleme dik olduğunu söylemektir. Bu, Şekil A6.3'te gösterilmiştir. Yeni vektör r'' ve vektör ürünü şu şekilde tanımlanır:

r'' = rx r'

r'' vektörü r ve r''yi içeren düzleme diktir ve büyüklüğü rr',sin(θ)'ye eşittir. r''nin bileşenleri şunlardır:

x'' = yz' - zy'

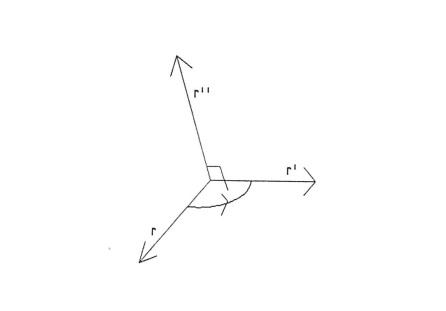
y'' = zx' - xz'

z'' = xy' - yx'

Vektör çarpımlarının matris çarpımları için de geçerli olan önemli bir yönü vardır; çarpım sırası önemlidir; rx r' çarpımı r' x r ile aynı değildir. Aslında

r' xr = -rx r'

r'' yönü, r'nin en küçük açıyla r'ye doğru bükülmesiyle elde edilir. Bunun saat yönünde bir dönüş olarak görüldüğü yön, r'' yönüdür.



*Şekil A5.3 Vektör çapraz çarpımı*

Vektörel çarpım karmaşıktır ancak bilgisayar grafiklerinde çok kullanışlıdır. Yüzeylere dik vektörler oluşturmak için kullanılır. Bunu daha sonra tartışacağız.

E.3.3. Yüzey Normal Vektörleri

Genellikle iki başka vektöre dik olan bir vektör oluşturmak gerekir. Bu, yüzey normal vektörlerinin ve koordinat dönüşümlerinin hesaplanmasında gerçekleşir. Bir yüzey normal vektörü durumunda amaç, yüzeye dik (dik açılarda) olan bir vektör oluşturmaktır.

Bu, önceki bölümde tartışıldığı gibi, yüzeyde yatan iki vektörün vektörel çarpımını oluşturmak anlamına gelir. Genellikle bu iki vektör bu şekilde sunulmaz, ancak poligon tepe noktası koordinat listelerinden oluşturulmaları gerekir. Bir dışbükey poligonun üç ardışık tepe noktasının

Pl(x1,y1,z1), P2(x2,y2,z2) ve P3(x3,y3,z3) ve bunların çevre boyunca saat yönünde ilerlediğini. Yüzeyden dışarıyı gösteren bir vektör elde etmek için bir çapraz çarpımda çarpılabilen iki vektör şunlardır:

r = (x3-x2)i + (y3-y2)j + (z3-z2)k

r' = (x2-x1)i + (y2-y1)j + (z2-z1)k

Biliyorum

r'' = rx r'

E.3.4. Temel Vektörler

Taban vektörleri, koordinat sisteminin eksenleri boyunca işaret eden birim vektörlerdir. Kartezyen koordinatlarda, i, j ve k "taban" vektörleridir. Her birinin büyüklüğü 1'dir, bu yüzden onları ayıran tek şey yönleridir.

E.4. Matrisler

Matrisler daha önceki bölümde ele alınmıştır. Bilgisayar grafiklerinde bir tür dönüşümü temsil ederler. Ele alınması en kolay olan matrisler, rotasyonla ilişkili olanlardır ve Ek 6'da daha ayrıntılı olarak ele alınmaktadır.

İki matrisi çarpma kuralı, bir matrisi ve bir vektörü çarpma kuralıyla aynıdır (önceki bölümde tartışıldığı gibi), burada vektör bir sütun ve üç satıra sahip bir matris olarak alınır. Vektöre ekstra sütunlar eklemek onu bir matris yapar ve üründe ekstra sütunlar üretir. Bir ürünün mümkün olması için ilk matriste ikinci matristeki satır sayısı kadar sütun olmalıdır.

Üç eksen x,y ve z etrafındaki dönüşü tanımlayan matrislerin hepsi üç satır ve üç sütuna sahiptir (homojen koordinatlarda olmadıkları sürece): 3x3 matrislerdir. Ayrı matrislerden belirli bir sırayla karmaşık bir dönüş oluşturma eylemi, matrisleri çarparak gerçekleştirilir. Buna matris birleştirme denir. Vektör çarpımında olduğu gibi, matris çarpımının sırası önemlidir: en sağdaki matris ilk dönüş ve en sola en yakın olan son dönüş olur.

E.5. Homojen Koordinatlar

Rotasyonların aksine, çeviriler ve perspektifler gibi belirli dönüşüm türleri 3 x 3 matrisler olarak yazılamaz ve vektörler üzerinde bir ürün olarak işlem yapılamaz. Birleştirme amacıyla tüm dönüşümleri eşit bir zemine oturtmak istendiğinden, tüm dönüşümleri çarpılabilen 4 x 4 matrislere dönüştürmek için homojen koordinatlar kullanılır.

Bu, ek boyutun her zaman 1 olduğu 4 boyutlu bir uzaya (gerçek uzay değil, sadece matematiksel bir kolaylık) geçmek anlamına gelir. Bu, tüm dönüşümleri 4 x 4 matrislere dönüştürmek için yeterli olan ek serbestlik derecesi sağlar. Aynı şekilde tüm vektörlerin dördüncü bir bileşeni, 1, olmalıdır. Bu dördüncü boyutu birliğe koymak, 4 boyutlu uzayda 1 kesişim noktasına sahip bir "düzlem" üzerinde çalıştığımız anlamına gelir. "Düzlem" normal 3 boyutlu uzaydır.

Ek F: Geometrik ve Koordinat Dönüşümleri

Bilgisayar grafiklerinde yaygın olarak kullanılan iki tür dönüşüm vardır: geometrik ve koordinat dönüşümleri. Kafa karıştırıcı olan şey, bunların aslında aynı şeyin iki yönü olması ve her iki yöntemle de aynı sonuca ulaşmanın mümkün olmasıdır. Ancak aklı başında kalmak için, bunları farklı olarak düşünmek, soruna bağlı olarak birini veya diğerini seçmek çok yardımcı olur. Aralarındaki ayrım ve bağlantı anlaşıldığında birçok akıllı kısayol mümkün hale gelir.

Siyah, özelliksiz duvarları olan bir odada dairesel bir halının ortasına yerleştirilmiş döner bir sandalyede oturduğunuzu hayal edin. Dışarıdan bir referans noktası olmadığından (gerçekte ne olduğunu hatırlamanın dışında), hareketsiz bir halı üzerinde sandalyeyi sağa döndürmek ile sandalyeyi sabit tutup halıyı sola döndürmek arasında ayrım yapmak mümkün değildir. Sandalyedeki gözlemci, sandalyenin ve halının aynı göreli hareketini görür ve halı desenine ilişkin görüşü her iki durumda da aynıdır. Ancak, tutarlı olan sandalyenin veya halının dönüş şemasını oluşturmaya dikkat etmeliyiz. Sola dönüşlerin pozitif, sağa dönüşlerin negatif olduğuna karar verelim. O zaman sandalyenin (gözlemcinin) pozitif dönüşünün halının (nesnenin) negatif dönüşüne eşdeğer olduğunu görebiliriz: bunların birbirinin tersi olduğu söylenir.

Şimdi resmi tanımlara geliyoruz. Gözlemciyi döndürmeye koordinat dönüşümü, nesneyi döndürmeye ise geometrik dönüşüm denir. Bilgisayar grafiklerinde bunların ikisini de yapmak istediğimiz birçok zaman vardır. Bir nesne dünya çerçevesinde hareket ettirildiğinde geometrik dönüşüme tabi tutulur. Dünyayı farklı bir bakış açısından görmek istediğimizde koordinat dönüşümü yapılmalıdır. Gözlemci bakış açısı yönelimini bir joystick aracılığıyla kontrol ettiğinde iki dönüşüm arasındaki bağlantıdan yararlanmak faydalıdır.

F.1. Koordinat Sistemleri ve Referans Çerçeveleri

Bir dereceye kadar bu terimler birbirinin yerine kullanılır. Çoğunlukla nesnelerin konumları ve köşeleri, dik açılarda üç x, y ve z ekseninden oluşan bir küme tarafından Kartezyen koordinatlarda belirlenir. Bu eksen kümesinin sıfırının konumuna koordinat sisteminin orijini denir. Bütün, çeşitli nesnelerin sonraki hareketini izlemek için bir referans çerçevesi oluşturur. Gördüğümüz gibi, iki tür hareket vardır: bir koordinat dönüşümü (gözlemci hareket ettiğinde) ve bir geometrik dönüşüm (bir nesne hareket ettiğinde). Nesne hareket ettiğinde, nesnenin kendisine bağlı referans çerçevesinin hareketini izleyerek ne olup bittiğini takip etmek en kolayıdır. Buna nesne çerçevesi adını verdik. Ana metinde nesneden dünyaya dönüşüm, seçilen dönüşler ve bu nesne çerçevesinin bir yer değiştirmesi ile yapılmıştır. Şimdi bunun tam olarak nasıl çalıştığını görebiliriz.

*Şekil A6.1 Bir nesnenin dönüşü*

udmage::figure-a6-1.jpg[genişlik='65%']

Nesneye kalıcı olarak tutturulmuş bir eksen kümesini hayal edin, böylece nesne hareket ettiğinde onlar da hareket eder. Basitlik açısından, Şekil A6.1'de gösterildiği gibi z ekseni etrafında θ açısıyla bir dönüşü ele alalım. Dönüşten sonraki koordinatları (x1,y1,z1,) (x,y,z)'den öncekilerle ilişkilendirmek için artık bir dönüşüm matrisine ihtiyaç vardır. Bu şemanın güzelliği, bu matrisi taban vektörlerine ne olduğunu gözlemleyerek oluşturabilmemizdir. Unutmayın, taban vektörleri x, y ve z eksenleri boyunca işaret direkleri gibi işaret eden birim vektörlerdir (boyut 1). Dönüşten önceki taban vektörleri i, j ve k'dir ve dönüşten sonraki taban vektörleri i1, j1 ve kl'dir. Şekle baktığımızda bunlar arasındaki ilişkileri görebiliriz:

i1 = cosθ.i + sinθ.j

j1 = -sinθ.i + cosθ.j

k1 = k

taban vektörleri için bir dönüşüm matrisine yol açar:

| cosθ sinθ 0 |

| |

| -sinθ cosθ 0 |

| |

| 0 0 1 |

Şimdi bu matris olduğu haliyle (x,y,z) koordinatlarını (x1,y1,z1)'e dönüştürmek için kullanılamaz, ancak ilginçtir ki tersi kullanılabilir. Neyse ki, saf bir rotasyonun tersi, satırları ve sütunları değiştirerek (transpoze ederek) elde edilir. Teknik dilde, bir rotasyonun tersi, onun transpozudur. Bunu yapmak matrisi verir:

| cosθ -sinθ 0 |

| |

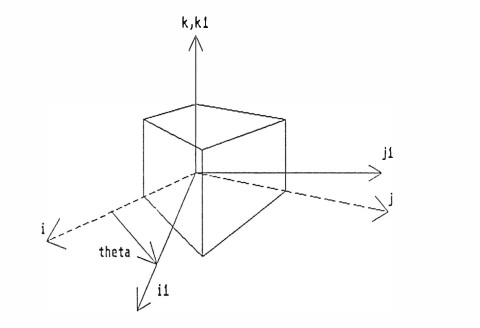
| sinθ cosθ 0 |

| |

| 0 0 1 |

böylece, örneğin, 90 derecelik bir dönüşte, (0,1,0) noktası (0,0,1) noktası olur ve (0,0,1) noktası (0,-1,0) olur. Yani bir nesneyi yeni bir yönelime döndürmenin bir yolunu bulduk: bu yeniden yönelimi nesne taban vektörleri üzerinde gerçekleştirin ve sonucu orijinal taban vektörleri açısından ifade edin: sonra matrisi transpoze ederek koordinat dönüşüm matrisini üretin.

Orijinal matris herhangi bir şey için kullanılabilir mi? Evet. Şu haliyle, transpoze edilmeden önce bir koordinat dönüşümüdür. Nesneyi sabit bırakıp sadece referans çerçevesini döndürürsek, nesne koordinatlarının yeni döndürülmüş çerçevede nasıl göründüğünü hesaplamak için dönüşümü verir. Bu, Şekil A6.2'de gösterilmiştir. Bu nedenle 90 derecelik dönüşte, (0,1,0) tepe noktası (0,0,-l) konumunda görünür ve (0,0,1) tepe noktası döndürülmüş çerçeveden bakıldığında (0,1,0) konumunda görünür. Bu dönüşlerin her ikisinde de, sırasıyla nesnenin ve referans çerçevesinin, dönüş yönünün pozitif olduğunu unutmayın.



*Şekil A6.2 Bir referans çerçevesinin dönüşü*

Şimdi döner sandalyedeki gözlemci ve halı ile ilgili nitel tartışmanın matematiksel olarak ifade edildiğini görebiliriz. Nesnenin pozitif dönüşünden sonra koordinatlarını hesaplayan dönüşüm şudur:

| cosθ -sinθ 0 |

| |

| sinθ cosθ 0 |

| |

| 0 0 1 |

ve referans çerçevesi pozitif yönde hareket ettirildikten sonra sabit nesnenin yeni görünür koordinatlarını hesaplayan dönüşüm şudur:

| cosθ sinθ 0 |

| |

| -sinθ cosθ 0 |

| |

| 0 0 1 |

İkisi de pozitif bir dönüş içerdiğinde farklıdırlar ancak referans çerçevesi (sandalye) negatif olarak döndürüldüğünde aynı olurlar. O zaman θ açısı negatiftir ve sin(-θ) = -sinθ ancak cos(-θ) = cosθ olduğundan sinθ içeren terimler işaret değiştirir ancak cosθ içerenler işaret değiştirmez.

Bu, referans çerçevesini bir yöne döndürmenin, nesneyi diğer yöne döndürmekle aynı göreli hareketi vereceği gerçeğini yeniden ifade ediyor.

Ek G: Program Yapısı

Bu ek, yeni dosyaların nerede tanıtıldığını görebilmeniz için her bölümün dosya içeriğini gösterir. Soldaki dosya ana kontrol dosyasıdır ve sağındaki dosyalar *içerdiği* dosyalardır . Bunların her biri için, altta parantez içindeki dosyalar *eklenen* dosyalardır.

G.1. Bölüm 3

polydraw.s core\_00.s bss\_OO.s systm\_00.s eşitler.s data\_00.s

G.2. Bölüm 4

clipframe.s core\_01.s bss\_01.s systm\_00.s eşitler.s data\_01.s

(çekirdek\_00.s) (bss\_00.s)

G.3. Bölüm 5

bakış açısı.s core\_02.s bss\_02.s systm\_00.s data\_01.s data\_02.s

(core\_01.s) (bss\_01.s) (data\_00.s) eşittir.s

G.4. Bölüm 6

otranw.s core\_03.s bss\_03.s sistem\_00.s data\_03.s data\_01.s

(çekirdek\_02.s) (bss\_02.s) (veri\_02.s) veri\_01.s

G.5. Bölüm 7

illhide.s core\_04.s bss\_04.s sistem\_00.s data\_03.s data\_01.s

(çekirdek\_03.s) (bss\_03.s) (veri\_02.s) veri\_00.s

veri\_04.s

G.6. Bölüm 8

trnsfrms.s core\_05.s bss\_05.s sistem\_00.s data\_03.s data\_00.s

(çekirdek\_04.s) (bss\_04.s) (veri\_02.s) veri\_05.s

G.7. Bölüm 9

wrld\_scn.s core\_07.s bss\_06.s sistem\_00.s data\_03.s data\_00.s

(çekirdek\_05.s) (bss\_05.s) (veri\_02.s) veri\_05.s

G.8. Bölüm 10

wrld\_scn.s core\_07.s bss\_07.s sistem\_01.s data\_00.s data\_06.s

(çekirdek\_06.s) (bss\_06.s) (sistem\_00.s) (veri\_02.s) (veri\_03.s)

veri\_07.s (veri\_05.s)

veri\_08.s

eulr\_scn.s çekirdek\_08.s

(çekirdek\_07.s)

Sürüm 5.4.1  
Son güncelleme 2019-06-25 21:06:24 UTC