DÜZCE ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

YÜKSEK LİSANS TEZİ

BİLGİSAYAR GRAFİKLERİ İÇİN FPGA TABANLI ÜÇGEN DOLDURMA MODÜL DİZAYNI

Emel OK

ELEKTRİK EĞİTİMİ ANABİLİM DALI

EYLÜL 2011
DÜZCE

DÜZCE ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

YÜKSEK LİSANS TEZİ

BİLGİSAYAR GRAFİKLERİ İÇİN FPGA TABANLI ÜÇGEN DOLDURMA MODÜL DİZAYNI

EMEL OK

ELEKTRİK EĞİTİMİ ANABİLİM DALI

EYLÜL 2011
DÜZCE

TEZ BİLDİRİMİ

Tez içindeki bütün bilgilerin etik davranış ve akademik kurallar çerçevesinde elde edilerek sunulduğunu, ayrıca tez yazım kurallarına uygun olarak hazırlanan bu çalışmada bana ait olmayan her türlü ifade ve bilginin kaynağına eksiksiz atıf yapıldığını bildiririm.

Emel OK

ÖNSÖZ

Yüksek lisans öğrenimim sırasında ve tez çalışmalarım boyunca gösterdiği her türlü destek ve yardımdan dolayı çok değerli hocam Yrd. Doç. Dr. İbrahim ŞAHİN'e teşekkür eder ve en içten minnet duygularımı sunarım.

EYLÜL 2011 **Emel OK**

İÇİNDEKİLER

Sayfa

ŞEKİL LİSTESİ

Sayfa

ÇİZELGE LİSTESİ

Sayfa

SEMBOL LİSTESİ

C1, …, C3, Cy1, …, Cy3	: yarı kenar fonksiyonu sabitleri
Cx1, …, Cx3	: yatay tarama için başlangıç değerleri
Dx12	: x_1 ile x_2 fark değeri
Dx23	: x_2 ile x_3 fark değeri
Dx31	: x_3 ile x_1 fark değeri
Dy12	: y_1 ile y_2 fark değeri
Dy23	: y_2 ile y_3 fark değeri
Dy31	: y_3 ile y_1 fark değeri
$\mathbf{T_{MODÜL}}$	: modüllerin çalışma süreleri
y	: koordinat düzleminde *y* ekseninde bir değer
y1	: üçgen verilerinde üçgenin birinci köşesinin *y* koordinatı
y2	: üçgen verilerinde üçgenin ikinci köşesinin *y* koordinatı
y3	: üçgen verilerinde üçgenin üçüncü köşesinin *y* koordinatı
x	: koordinat düzleminde *x* ekseninde bir değer
x1	: üçgen verilerinde üçgenin birinci köşesinin *x* koordinatı
x2	: üçgen verilerinde üçgenin ikinci köşesinin *x* koordinatı
x3	: üçgen verilerinde üçgenin üçüncü köşesinin *x* koordinatı

EK LİSTESİ

Sayfa

KISALTMALAR

ASIC	: Uygulamaya Özel Tümdevre
CLB	: Yapılandırılabilir Mantıksal Bloklar
CPU	: Merkezi İşlem Birimi
DSP	: Sayısal Sinyal İşlemcisi
EDA	: Elektronik Tasarım Otomasyonu
F-CCMs	: FPGA Tabanlı Özel Bilgi İşlem Makinaları
FPGA	: Alanda Programlanabilir Kapı Dizileri
FSM	: Sonlu Durum Makinesi
IOB	: Giriş/Çıkış Blokları
LUT	: FPGA İçinde Yer Alan Danışma Tablosu
PCI	: Bilgisayar Çevre Birim Arayüzü
PNN	: Olasılıksal Sinir Ağı
RC	: Yeniden Konfigüre Edilebilir Bilgisayar
RICS	: Azaltılmış Komut Kümesi Bilgisayarı
RTL	: Register Dönüşüm Lojiği
VHDL	: Çok Yüksek Hızlı Tümleşik Devre Donanım Tanımlama Dili

BİLGİSAYAR GRAFİKLERİ İÇİN FPGA TABANLI ÜÇGEN DOLDURMA MODÜL DİZAYNI
(Yüksek Lisans Tezi)

Emel OK

DÜZCE ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
Ağustos 2011

ÖZ

Bilgisayar grafiklerinde poligon rendering (giydirme-kaplama) işlemi yüzeyleri matematiksel olarak poligonlarla tanımlanmış animasyon objelerinin renklendirilme işlemidir. Grafik sahnesindeki nesne sayısı ve bu nesneleri tanımlama kullanılan poligon sayısı arttıkça render işlemi içini gerekli olan CPU zamanıda katlanarak artmaktadır ve genel amaçlı bilgisayarlar rendering işleminde yetersiz kalmaktadırlar. Bu durum karşısında gelişmiş grafik kartlarının, grafik işlemler için tasarlanmış özel amaçlı bilgisayarların, daha hızlı işlemcilerin ya da paralel işlemcilerin kullanılması gibi farklı yaklaşımlar geliştirilmiştir. Ancak bu yöntemlerde yüksek maliyetler gerektirmekte ve her zaman istenen performans elde edilememektedir. Bu çalışmada yukarıda bahsi geçen yaklaşımlara bir alternatif olarak rendering işlemini gerçekleştimek üzere Alanda Programlanabilir Kapı Dizileri (Field Programmable Gate Array (FPGA)) yongaları üzerinde çalışabilecek tam sayı tabanlı iki farklı donanım modülü tasarlanmıştır. Birinci modül Yarı Alan denkleminden yararlanılarak oluşturulmuştur ve koordinatları verilen verilen üçgeni tek renk ile doldurmaktadır. İkinci modül ise Doğru Denkleminden yararlanılarak oluşturmuştur ve üçgenin içini köşe noktaların renk bilgisine göre tonlama yaparak doldurmaktadır. Tasarlanan modüller, rastgele oluşturulmuş test üçgenleri kullanılarak test edilmiştir. Modüllerin verilen üçgenleri render etme süreleri genel amaçlı bilgisayralar üzerinde çalışan ve C++'da oluşturulan yazılım versiyonları ile karşılaştırılmıştır. Sonuçta genel amaçlı bilgisayarlara göre birinci modülün 31 kata kadar, ikinci modülün ise 17 kata kadar değişen oranlarda daha hızlı rendering yaptığı görülmüştür.

Bilim Kodu :
Anahtar Kelimeler : FPGA, Bilgisayar Grafikleri, Yüzey Giydirme, Donanım Modülü
Sayfa Adedi : 83
Tez Yöneticisi : Yrd. Doç. Dr. İbrahim ŞAHİN

AN FPGA BASED TRIANGLE RENDERING MODULE DESIGN FOR COMPUTER GRAPHICS

(M.Sc. Thesis)
Emel OK

DUZCE UNIVERSITY
INSTITUTE OF SCIENCE AND TECHNOLOGY
August 2011

ABSTRACT

In computer graphics, rendering is the process of coloring the animation objects whose sufaces are mathematically defined with poloygons. As the number of objects on a scene and the number of poligons used to define these objects increase, the CPU time requirement of the rendering process grows exponentially and general purpose computers become insufficient. Several approaches such as using exhanced graphics cards, specially degined computers, or parellel computers were developed. But these approaches are not cost effective and sometimes do not yield the desired performance. In this research work, as alternatives to the above mentioned approaches, two separate hadware modules were designed. The modules were designed to be used with Field Programmable Gate Array (FPGA) and can process fixed-point data. The first module uses the half-space property of the line equation. This module is able to fill a given triangle with a solid color. The second module uses the line equation itself. This module is able to fill inside a given triangle using the given intensity values at the corners of the triangle. Each pixel's color value is calculated according to pixel's distance to the corners. The modules were tested using randomly generated test triangles. Modules' rendering times were compared to redering softwares' CPU times. The rendering softwares were coded in C++ and were running on general purpose computers. The results showed that the first module renders the test triangles up to 31 times faster and the second module renders the test triangles up to 17 times faster.

Science Code :
Key Words : FPGA, Computer Graphics, Rendering, Hardware Module
Page Number: 83
Adviser : Assist. Prof. Dr. İbrahim ŞAHİN

1. GİRİŞ

Bilgisayar grafiklerinde "rendering" (poligon giydirme, kaplama), üçgenlerle matematiksel olarak modellenmiş animasyon objelerinin giydirilme işlemidir. Bilgisayar grafiklerinin kullanıldığı birçok alanda rendering işlemine ihtiyaç duyulmaktadır. Render edilen nesnenin görünümünün gerçekciliğini artırmak için render edilecek üçgenlerin üç boyutlu (3B) ortamda kamera ve ışık kaynağına göre konumu, aydınlanma miktarı ve diğer objelerden oluşan yansımalar dikkate alınmaktadır (Fender ve Rose, 2003). Rendering işlemi yazılım ve donanım olarak farklı şekillerde gerçeklenmiştir. Yazılımsal çözümler genellikle zaman problemi olmayan ve çok hassas hesaplamaya gerek duyulmayan durumlarda kullanılmaktadır. Donanımsal çözümler ise daha hızlı hesaplama yapılması gereken yerlerde tercih edilmektedir.

Bilgisayar grafiklerinde sahnede kullanılan nesneleri modellemede kullanılan üçgenlerin boyutları nekadar küçük olursa görüntü kalitesi de o kadar yüksek olur. Bu durumda da nesne modellemede kullanılan üçgen sayısında ciddi bir artış olmaktadır. Ancak grafik sahnesindeki nesne sayısı ve bu nesneleri tanımlamada kullanılan üçgen sayısı arttıkça, grafik sahnesinin render edilme süresi saatler alabilmektedir (Anon, 20 Şubat 2010). Bu durumda genel amaçlı bilgisayarlar yetersiz kalmakta ve grafik sahnesinin hesaplanması çok uzun süreler almaktadır. Bu istenmeyen durum farklı yaklaşımlar geliştirilerek çözülmeye çalışılmıştır. Bu yaklaşımlardan bazıları; gelişmiş grafik kartların, grafik işlemler için tasarlanmış özel amaçlı bilgisayarların (Anon, 17 Mart 2009), daha hızlı işlemcilerin ya da paralel işlemcilerin kullanılması olarak sıralanabilir. Ancak bu çözümlerin bazı dezavantajları da söz konusu olmaktadır. Bu dezavantajlar; özel olarak tasarlanan kartlarda herhangi bir tasarım veya üretim hatasının telafi edilememesi ve yeniden tasarım sürecinin fazla zaman kaybına sebep olması, özel tasarlanmış bilgisayarlar ve paralel işlemcili bilgisayarların kullanımının yüksek maliyetli olması ve her zaman istenen performansın elde edilememesi olarak sıralanabilir. Bu yaklaşımlara bir alternatif ise Alanda Programlanabilir Kapı Dizileri (Field Programmable Gate Array (FPGA)) tabanlı çözümlerdir. Literatürde FPGA'ler

için tasarlamış genel amaçlı rendering modülleri görmek mümkündür (Fender ve Rose, 2003), (Beeckler ve Gross, 2005), (Majer ve diğ., 2004)

Çalışmanın Amacı; yukarıda sayılan çözümlere bir alternatif olarak üçgen rendering işlemlerini daha hızlı gerçekleştirmek üzere FPGA yongaları üzerinde çalışabilecek tam sayı tabanlı donanım modülleri tasarlamak ve FPGA' lerin sağladığı bir çok avantajdan yararlanmaktır. FPGA'lerin sağladığı bu avantajlar;

- Daha ucuz (paralel bilgisayarlardan, süper bilgisayarlardan ve özel tasarlanmış grafik kartlarından daha düşük maliyetlere sahiptirler);
- Daha hızlı (basit ve hızlı bir tasarım süreci, tasarımdan sonra pratik üretim süreçleri gerektirdiğinden FPGA yongalarına kısa sürede yüklenerek kullanıma çabuk hazır olmakta ve normal bilgisayarlarla karşılaştırıldığında daha yüksek performans göstermektedirler);
- Daha esnek (yeniden programlanabilme özelliği ile çok kısa bir sürede ve defalarca programlanabilerek herhangi bir zarara uğranılmadan tasarım yenilenebilmektedirler) olarak özetlenebilir.

Bu hedefler doğrultusunda Rendering işlemini gerçekleştirebilmek için FPGA üzerinde çalışabilen iki farklı donanım modülü tasarlanmıştır. Birinci modül Yarı Alan denkleminden, bir doğru parçasının çizim alanını iki yarıya bölmesi durumundan, yararlanılarak oluşturulan Rendering algoritmasını gerçekleyecek şekilde tasarlanmıştır. İkinci modül ise Doğru Denkleminden, bir doğru denkleminde geçerli y değerine karşılık gelen x değerinin bulunabilmesi durumundan, yararlanılarak oluşturulan Rendering algoritmasını gerçekleyecek şekilde tasarlanmıştır.

Tasarlanan modüller, rastgerel oluşturulmuş değişik büyüklüklerdeki test üçgenleri kullanılarak test edilmiş ve modüllerin ürettiği sonuçların doğrulaması yapılmıştır. Aynı üçgenler farklı özellikteki genel amaçlı bilgisayarlarla da render edilerek, modüllerin rendering süreleri bilgisayarların rendering süreleriyle karşılaştırılmıştır. Sonuçta test üçgenlerini, üçgen boyutuna oranla ve C++ kodun test edildiği genel amaçlı bilgisayarlara göre değişmekle beraber, birinci modülün 31 ile 1.5 arasında, ikinci modülün ise 17 ile 0.5 arasında daha hızlı render ettiği görülmüştür.

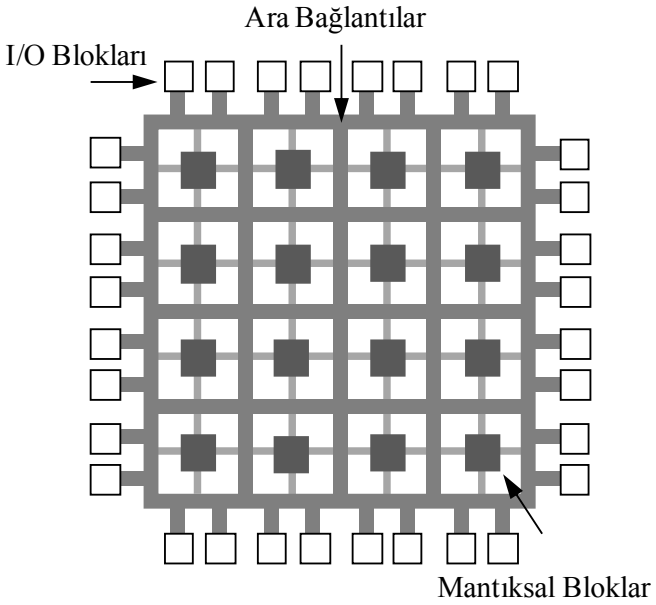
Çalışmanın ikinci bölümünde FPGA yongaları ve RC (Yeniden Konfigüre Edilebilir Bilgisayar) sistemler, bilgisayar grafikleri ile hedeflenen rendering yöntemi kısaca özetlenmiştir. Üçüncü Bölümde, tasarlanan modüller anlatılmıştır. Dördüncü Bölümde, yapılan test çalışmaları ve bu çalışmalardan elde edilen sonuçlar sunulmuştur. Beşinci Bölümde ise sonuçlar değerlendirilmiştir.

2. ÖN BİLGİ VE DİĞER ÇALIŞMALAR

2.1. FPGA YONGALARI

FPGA'lar üretim sonrası müşteri yada tasarımcı tarafından herhangi bir mantıksal fonksiyonu gerçekleyebilmek için yapılandırılabilen entegre devrelerdir. Bu yapılandırma genellikle, uygulamaya özgü bir entegre devre (ASIC) için kullanılana benzer, bir donanım tanımlama dili (HDL) kullanılarak yapılır (Anonim, 11 Ağustos 2011). Böylece FPGA'ler tasarım ya da yükleme aşamasında herhangi bir hata yapılması söz konusu olduğunda tekrar tekrar tasarlanıp yapılandırılabilmektedirler.

FPGA'lar tek yonga yapılardır. Bu tek yonga yapılar genelleştirilmiş sayısal kaynakları, programlanabilen bağları, mikroişlemcileri, çarpıcıları, hafızaları ve diğer birçok genel çekirdek yapısını içerebilmektedir. İçerisindeki bağlantılar, hafızasına gerekli bit akıntısı yüklenerek yapılandırılabilen mantık kapıları dizisidir (Koca, 2007). Şekil 2.1'de görüldüğü gibi FPGA yongalarına ait genel yapı üç temel bileşenden oluşur. Bunlar: yapılandırılabilir mantıksal bloklar, giriş çıkış blokları ve ara bağlantılardır.



Şekil 2.1: FPGA çiplerinin genel yapısı (Anon, 12 Ekim 2009)

2.1.1. Yapılandırılabilir Mantıksal Bloklar (Configurable Logic Blocks (CLB))

Mantıksal fonksiyonların oluşturulabildiği Look-up table (LUT) ve lojik kapılardan oluşan donanım bloklarıdır. Bu bloklar karmaşık kombinasyonel işlemleri ya da VE, VEYA gibi basit mantık kapıları gerçekleştirmek için yapılandırılabilir (Anonim, 11 Ağustos 2011). Böylece oluşturulmak istenen mantıksal devre için oldukça fonksiyonel ve esnek eleman imkanları sunmaktadır. CLB mimarisinin esnek ve simetrik olması, uygulamaların kolaylıkla yerleştirilebilmesine ve yönlendirilebilmesine olanak tanır. FPGA çip modeline ve üretici firmaya göre farklılıklar gösterir (Koyuncu, 2008).

2.1.2. Giriş Çıkış Blokları (Input/Output Blocks (IOB))

İhtiyaç duyulan sinyallerin alınması yada aktarılması için kullanılır. IOB'ler yonganın iç sinyal hatları ile pinleri arasında programlanabilir bir arabirimdir. Bu sayede IOB'ler giriş, çıkış ya da çift yönlü olarak programlanabilir. FPGA yongasının türüne göre bir yongadaki IOB sayısı (pin sayısı) 1000'li sayılara ulaşabilmektedir(Koyuncu, 2008).

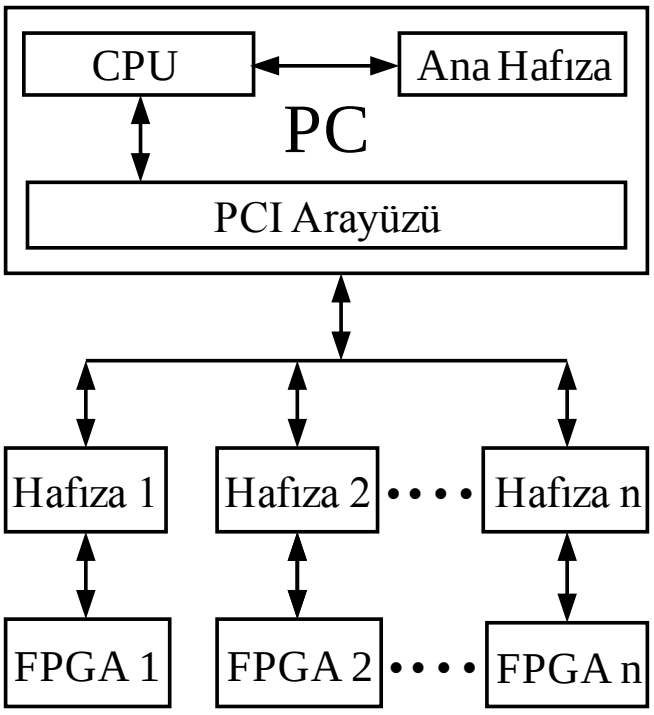
2.1.2. Ara Bağlantılar (Interconnections)

Bu birimler hem CLB'ler arasında hem de CLB'ler ile IOB'ler arasında bağlantıları şekillendirmede kullanılan programlanabilir anahtarlardır (Koyuncu, 2008).

2.2. FPGA TABANLI ÖZEL AMAÇLI BİLGİSAYARLAR

Temel olarak donanım ve yazılımı bir araya getiren özel veri işleme platformları olan FPGA tabanlı Özel Amaçlı Bilgisayarlar (FPGA-based Custom Computing Machines (F-CCMs)), ayrıca yeniden yapılandırılabilir özelliğine sahip olduklarından dolayı Tekrar Yapılandırabilir Bilgisayar (Reconfigurable Computing (RC)) sistemleri olarak da bilinirler (Sahin ve Gloster, 2005a). Şekil 2.2'de genel yapısı görülen RC sistemleri bir adet genel amaçlı bilgisayar ve üzerinde bir ya da daha fazla FPGA ve hafıza yongaları bulunduran elektronik kart veya kartlardan oluşurlar (Sahin ve Gloster, 2000), (Sahin ve Gloster, 2005b), (Sahin ve Gloster, 2001). RC sistemlerde *n* adet FPGA yongası ve bunlara ait yerel hafızalar bulunur. PCI ara yüzü, FPGA kartının bilgisayar ile veri iletişimini sağlamaktadır (Koyuncu, 2008). Genel amaçlı işlemciler sayesinde programlanabilme özelliliğine, FPGA yongaları sayasesinde ise hızlı tasarım ve üretim özelliğine sahip sistemlerdir (Rincon ve Teres, 1998). F-CCM'lerde uygulamaların

yüksek işlemci gücü gerektiren bölümleri, özel olarak tasarlanmış donanım modülü kullanılarak FPGA yongaları üzerinde çalıştırılarak programın genelinin daha hızlı çalışması sağlanır(Koyuncu, 2008). Çeşitli algoritmaların uygulandığı F-CCM'ler ve genel amaçlı bilgisayarlar karşılaştırıldığında, F-CCM'lerin daha düşük çalışma zamanları verdiği görülmüştür (Sridharan ve Priya, 2007), (Bishop ve Kelliher, 2003). FPGA kartlarının çalışma prensibi ise şu şekildedir: özellikle FPGA çiplerine modülün konfigürasyon bilgisi yüklenerek konfigüre edilir. Ardından FPGA hafızasına PCI üzerinden işlenecek veriler depolanarak FPGA de konfigüre edilen modüle bir başlat sinyali gönderilerek modülün hafızadaki veriyi işlemesi ve yeni veriler üretmesi sağlanır. Veriler işlenirken, ana bilgisayar diğer işlerine devam eder. FPGA yongaları verileri işlemeyi bitirince, bilgisayara bir kesme sinyali göndererek işlenmiş verinin hazır olduğunu bildirmektedir. Böylece bilgisayarın PCI üzerinden FPGA kartı üzerindeki hafızalara erişerek işlenmiş veriyi almasını sağlar (Sahin ve Gloster, 2005b), (Koyuncu, 2008).



Şekil 2.2: RC sistemlerin genel yapısı (Koyuncu, 2008)

2.2.1. FPGA Yongaların Avantajları

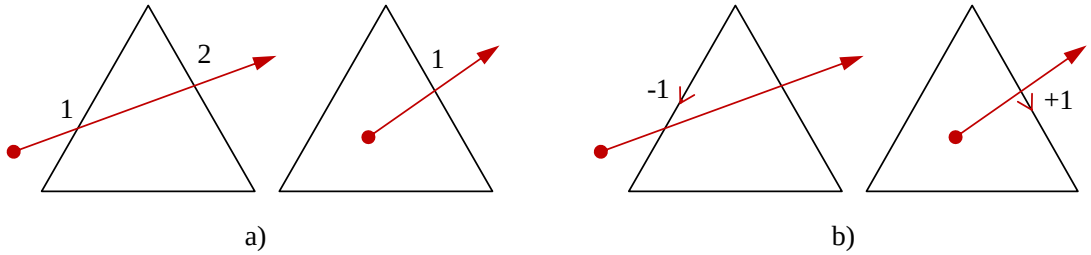
İlk üretimde piyasaya hızlı bir şekilde sunulabilmesi, tasarım sırasında esneklik sağlayabilmesi ve paralel işlem yapabilme özellikleriyle FPGA yongaları oldukça avantajlıdırlar. Uygulamaya Özel Tümdevre (Application Spesipic Integreted Circuit (ASIC)) yongaları, ancak uzun süren çeşitli tasarım ve üretim aşamaları sonrasında kullanıma hazır hale gelebilmektedir. FPGA yongaları kullanılarak, bu uzun üretim süresi tasarlanan ürünün FPGA yongalarına yüklenmesiyle tek aşamada ve oldukça kısa bir sürede gerçekleştirilebilmektedir. Ayrıca tasarımda yapılabilecek herhangi bir hata sonucu ASIC yongaları kullanılamaz hale gelirken FPGA yongaları yeniden programlanabilme özelliği ile çok kısa bir sürede ve herhangi bir zarara uğranılmadan yeniden programlanabilmektedir. FPGA yongalarının, boyutlarına ve tasarıma bağlı olarak, programlanma süresi mili saniyeler civarındadır. FPGA yongalarının tekrar programlanabilme özelliği sınırsızdır ve 550MHz saat hızlarına kadar çıkabilmektedirler (Anon, 09 Ağustos 2011), (Anon, 10 Ağustos 2011). Karmaşık ve fazla kapasiteye gerek duyan lojik devre tasarımlarında kullanılabilen FPGA yongaları, tek bir problemin çözümü için tasarlandıklarından yazılıma oranla çok daha hızlı çalışmaktadırlar. Ayrıca tek bir çip içerisinde tasarlanan modülün boyutuna bağlı olarak birden fazla modül yerleştirilebilmesi ve birden fazla FPGA çipi birbiri ile paralel olarak çalıştıralabilmesi ile yazılıma göre çok yüksek hız kazançları elde edilebilmektedir(Koyuncu, 2008).

2.3. BİLGİSAYAR GRAFİKLERİNDE RENDERİNG

Rendering (poligon giydirme, kaplama, doldurma) işlemi, Şekil 2.3' te görüldüğü gibi yüzeyi, köşe nokta koordinatları belli olan poligonlarla matematiksel olarak modellenmiş grafik objelerinin giydirilme işlemidir. Grafik objeleri modellenirken matematiksel hesaplamaları en aza indirmek için genellikle üçgenler kullanılır. Bu modellemede kullanılan üçgen boyutları ne kadar küçük olursa Rendering işlemi sonrasında görüntü kalitesi de o kadar yüksek olmaktadır. Rendering işlemi bilgisayar grafiklerinin kullanıldığı mimari tasarım yazılımlarından, bilgisayar oyunlarına, simulatörlerden, sinema ve televizyon gibi sektörlere kadar geniş bir alanda kullanılmaktadır.

Şekil 2.3: Üçgenlerle matematiksel olarak modellenmiş örnek nesne (Anon, 10 Eylül 2010)

Poligon doldurma işlemi için geçerli pixelin poligon içinde olup olmadığını kontrol etmek üzere hali hazırda geliştirilmiş birçok yöntem bulunmaktadır. Bunlardan sıklıkla kullanılanları Şekil 2.4 a'da verilen tek-çift kuralı (odd-even rule) ve Şekil 2.4 b'de görülen sıfır olmayan dolanma sayısı kuralı (nonzero winding number rule)'dır. Her iki yöntem için de çizim alanında keyfi bir P noktası belirlenir ve bu noktadan başlayarak sonsuza giden bir doğru çizilir. Tek çift kuralında, eğer bu doğrunun Rendering işlemi yapılan poligonun kenarlarını kestiği nokta sayısı tek adet ise nokta poligonun içinde, çift adet ise nokta poligonun dışında demektir. Sıfır olmayan dolanma sayısı kuralında ise; çizim alanında bulunan poligonun köşe noktalarından soldan sağa doğru ilerlenerek aynı köşeye ulaşıldığında arada kalan bölgenin değeri +1 kabul edilir. Sağdan sola doğru ilerlenildiğinde ise arada kalan bölgenin değerinin -1 olduğu kabul edilir. Çizim alanında herhangi bir noktadan sonsuza bir doğru çizildiğinde, doğrunun poligon kenarlarını kestiği nokta için kenarlara ait değerler toplanır. Son olarak toplam dolanma sayısı 0 ise nokta poligonun dışında, değilse içindedir (Hearn ve Baker, 2003).

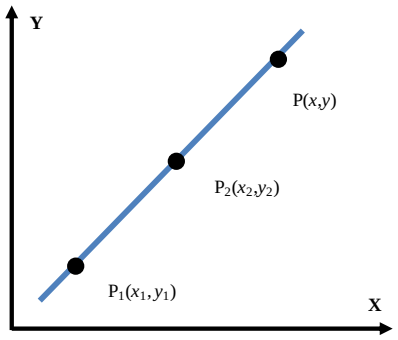


Şekil 2.4: a) Tek-çift kuralı, b) Sıfır olmayan dolanma sayısı kuralı

Çalışmamızda yukarıda anlatılan yöntemlere alternatif, doğru denkleminden yararlanılarak iki farklı yöntem oluşturulmuştur. Bu yöntemler, Yarı Alan Denklemi ve Doğru Denklemidir.

2.3.1. DOĞRU DENKLEMİ

Çalışmada kullanılan her iki Rendering yöntemi de doğru denkelemi temel alınarak oluşturulduğundan bu bölümde öncelikle doğru denklemi açıklanacaktır. Şekil 2.5'te verilen ve iki noktası bilinen, koordinatları (*x1*, *y1*), (*x2*, *y2*) olan, bir doğru parçasının denklemi Denklem 2.1'de görüldüğü gibidir. Bu denkelem doğru üzerindeki herhangi bir (*x*, *y*) ikilisi için geçerlidir.



Şekil 2.5: İki noktası bilinen doğru parçası

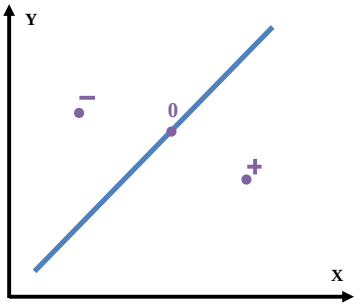
$$\frac{y_2-y_1}{x_2-x_1} = \frac{y-y_2}{x-x_2} \tag{2.1}$$

Yukarıdaki denklemden yararlanılarak iki farklı rendering yöntemi oluşturulmuştur. Bunlar Yarı Alan Kontrolü ile Rendering ve Doğru Denklemi Kontrolü ile Rendering algoritmalarıdır.

2.3.1.1. Yarı Alan Fonksiyonu Kullanarak Rendering

Denklem 2.1'de içler dışlar çarpımı işlemi yapıldıktan sonra Denklem 2.2 elde edilir. Doğru üzerindeki herhangi bir noktanın koordinat değeri (*x*,*y*) bu eşitliğe uygulandığında eşitlik 0 değerini verir.

$$(y_2 - y_1) * (x - x_2) - (y - y_2) * (x_2 - x_1) = 0 \qquad (2.2)$$

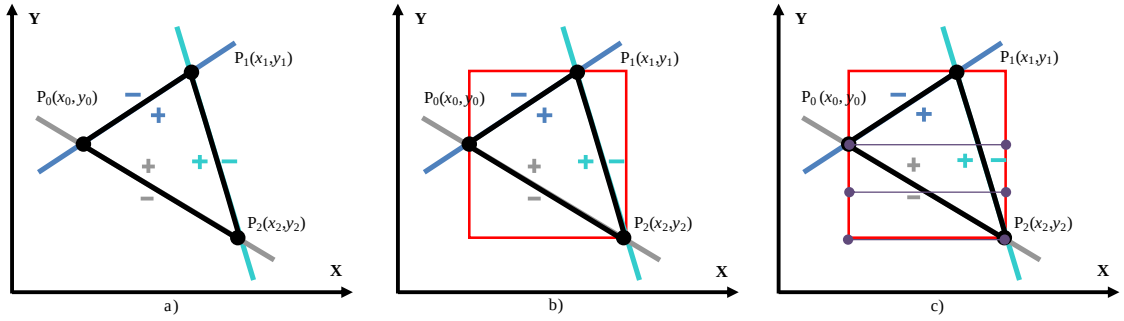


Şekil 2.6: Yarı Alanı İşlevi

Doğru dışındaki noktalar için ise eşitlik, seçilen noktanın konumuna göre (doğrunun sağında veya solunda olma durumuna göre) pozitif veya negatif bir sonuç verir. Dolayısıyla bir doğrunun ekranı iki yarıya böldüğü Şekil 2.6' da görülmektedir. Bu duruma yarı alanı işlevi denilmektedir. Bu işlev bir üçgenin kenarlarını oluşturan doğrulara tektek uygulanarak düzlem üzerinde seçilen herhangi bir noktanın üçgenin içinde olup olmadığı test edilebilir. Şekil 2.7 a'da üçgenin her bir kenarının, kendine göre tarama yapılan ekran alanını pozitif ve negatif olarak ikiye ayırdığı görülmektedir.

Yarı alan işlevi kullanılarak olşturulan algoritma mantığı şu şekildedir: Şekil 2.7 c' de görüldüğü gibi öncelikle render edilecek üçgenin içinde yeraldığı en küçük dikdörtgen alan (Bounding Rectangle) üçgenin köşe noktalarının en küçük ve en büyük *x* ve *y* değerleri kullanılarak belirlenir. Bunun amacı ekrandaki bütün pixellerin test edilmesinden ziyade sadece belli alandaki pixellerin teste tabi tutulması ve dolayısıyla işlem süresinin kısaltılmasıdır. Bounding rectangle bulunduktan sonra bu alan içindeki bütün pixeller Şekil 2.7 c'de görüldüğü gibi üçgenin kenarlarını oluşturan üç doğrunun denklemlerine göre sırayla test edilir. Eğer geçerli pixelin koordinatları ile yapılan test sonucunda üç denklemden de pozitif bir değer alınırsa bunun anlamı geçerli pixel üçgenin içindedir. Ancak denklemlerden herhangi birinden sıfır değeri diğerlerinden ise pozitif bir değer elde edilmesi halinde geçerli pixelin sıfır değerini veren kenar üzerinde

olduğu anlamına gelmektedir. Bu durumlarda geçerli pixelin ekranda bulunduğu alana ait hafıza adresi hesaplanarak bu hafızaya pixelin renk bilgisi yazılır. Ayrıca bu üç denklemden elde edilen sonuçlardan herhangi biri yada herhangi ikisinden elde edilen sonuçlar negatif ise bu geçerli pixelin üçgenin dışında olduğu anlamına gelir. Bu durumda ise herhangi bir işlem yapılmadan test edilmek üzere bir sonraki pixelin koordinatlarına gidilir.



Şekil 2.7: a) Yarı Alan, b) Üçgeni çevreleyen en küçük dörtgen (Bounding Rectangle), c) Yarı Alan İşlevi ile Bounding Rectangle'da yapılan tarama

Yukarıda anlatılan mantık çerçevesinde oluşturulan algoritma Şekil 2.8'de görüldüğü gibidir (Capens, 2009). Algoritmanın birinci bölümünde denklemlerde yer alan sabit *x* ve *y* farkları hesaplanmaktadır. İkinci bölümde bounding rectangle oluşturmak üzere üçgenin köşe nokta koordinatlarının en küçük ve en büyük *x* ve *y* değerleri tespit edilir. Üçüncü bölümde ise bounding rectangle içindeki ilk pixelin koordinat değerlerine göre üç doğru denklemi hesaplanır. Dördüncü bölümde ise Şekil 2.7 c'de görüldüğü gibi bounding rectangle sınırları içinde sırayla her pixel test edilmek üzere iç içe yeralan iki döngüye girilir. Döngü denklemler yatayda *y* farkları kadar azaltılıp, dikeyde *x* farkları kadar arttırılarak her pixel için üç denklemden döngü içerisinde sadece toplama çıkarma işlemleri yapılarak birer sonuç elde edilir. Bu sonuçlar içteki döngüde kontrol edilerek geçerli pixelin üçgenin içinde mi dışında mı olduğuna karar verilirek belirli renk değeri o pixelin ekran hafızasındaki adresine yazılır. Ardından dış döngüye geri dönerek bounding rectangle taraması bitene kadar bu işlemleri tekrar eder.

```
Render (float x1, float y1, float x2, float y2, float x3, float y3)
begin
    1. Delta değerlerini hesapla
    Dx12 = x1 - x2;  Dx23 = x2 - x3;  Dx31 = x3 - x1;
    Dy12 = y1 - y2;  Dy23 = y2 - y3;  Dy31 = y3 - y1;
    2. Sınırlayan dikdörtgeni kararlaştır
    minx = min(x1, x2, x3);  maxx = max(x1, x2, x3);
    miny = min(y1, y2, y3);  maxy = max(y1, y2, y3);
    3. Yarı kenar fonksiyonunun sabit parçalarını hesapla
    C1 = Dy12 * x1 - Dx12 * y1;
    C2 = Dy23 * x2 - Dx23 * y2;
    C3 = Dy31 * x3 - Dx31 * y3;
    Cy1 = C1 + Dx12 * miny - Dy12 * minx;
    Cy2 = C2 + Dx23 * miny - Dy23 * minx;
    Cy3 = C3 + Dx31 * miny - Dy31 * minx;
    4. Sınırlayan dikdörtgen boyunca tara
    for (y = miny; y < maxy; y++)
    begin
        Yatay tarama için başlangıç değeri
        Cx1 = Cy1;
        Cx2 = Cy2;
        Cx3 = Cy3;
        for (x = minx; x < maxx; x++)
        begin
            if (Cx1 > 0 and Cx2 > 0 and Cx3 > 0)
                colorBuffer[x] = 0x00FFFFFF;
            Cx1 -= Dy12;
            Cx2 -= Dy23;
            Cx3 -= Dy31;
        end for
        Cy1 += Dx12;
        Cy2 += Dx23;
        Cy3 += Dx31;
    end for
end
```

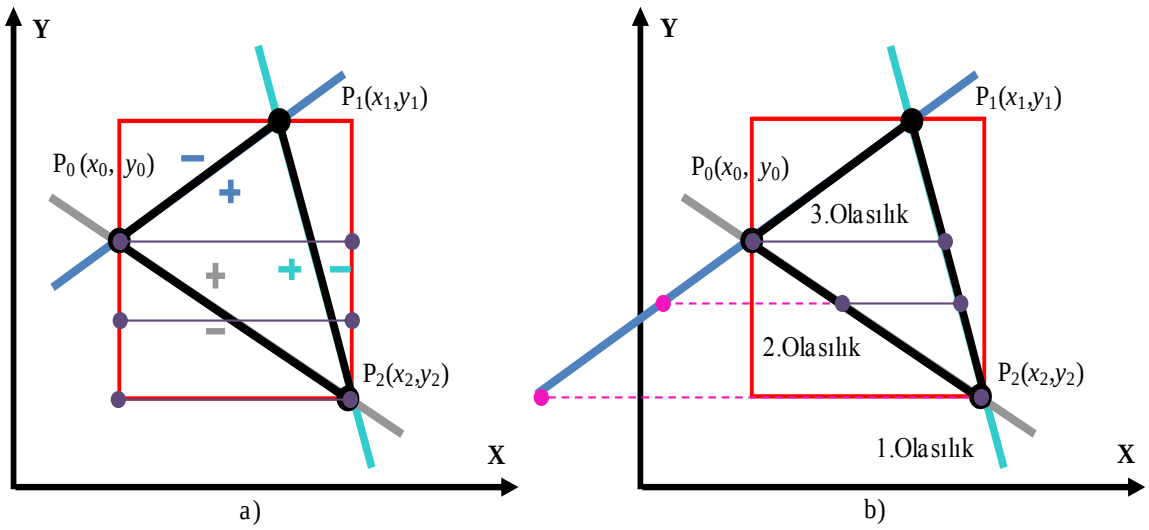
Şekil 2.8: Yarı Alan işlevi ile tarama algoritması (Capens, 2009)

2.3.1.2 Doğru Denklemi ile Rendering

2.1'de verilen doğru denkleminde yeralan *x* yalnız bırakılarak Denklem 2.3 edilmiştir. Bu denklem kullanılarak Doğru Denklemi adı altında bir Rendering algoritması oluşturulmuştur. Bu algoritmada Şekil 2.9 a'da görüldüğü gibi bounding rectangle içindeki bütün pixellerin sırayla taranması yerine Şekil 2.9 b'de görüldüğü gibi her satırda sadece üçgenin kenarları üzerinden başlayacak ve bitecek şekilde tarama yapılır. Bunun için geçerli *y* değerine karşılık gelen tüm yarı alan formüllerinden elde edilen *x* değişkenleri Denklem 2.3 kullanılarak hesaplanır. Şekil 2.9 b'de görüldüğü gibi bu durumda elde edilen *x* değerleri arasından sadece, üçgeni çevreleyen en küçük dörtgen (Bounding Rectangle) içerisinde yeralan *x* değerlerine ait pixellerin üçgenin üzerinde olduğundan bahsedilebilir. Dörtgen sınırları içerisinde Şekil 2.9 b' de görüldüğü gibi üç farklı olasılıkta ancak iki ya da üç *x* değeri elde edilebilir ve üçgenin şekline göre hangi

kenara ait doğru denkleminden elde edilecek bu *x* değerliri farklılık gösterecektir. Bu durumda ilk olasılık; sadece iki *x* değeri elde edilmesi ve bu değerlerin birbirine eşit olması durumudur. Bu bir köşe noktasını işaret etmektedir. İkinci olasılık; yine sadece iki *x* değeri elde edilmesi ancak bu değerlerin birbirine eşit olmaması durumudur. Bu durumda ise geçerli *y* değerine karşılık gelen iki kenarın varlığı söz konusudur. Üçüncü olasılık ise üç adet *x* değeri elde edilmesi ve bu üç değerinde dörtgenin içerisinde olması durumudur. Bu durumda değerlerden ikisi aynı fakat biri faklı olmak zorundadır ve geçerli *y* değerine karşılık gelen bir köşe bir kenar vermektedir. Bu işlemlerden sonra üçgenin üzerinde olduğu tespit edilen *x* değerlerinin en küçük ve en büyüğü yapılacak her satır taraması için başlangıç ve bitiş değerleri olarak belirlenir. Böylece sadece üçgenin içinde tarama yapılması sağlanmış olur. Bir üçgeni çevreleyen en küçük dörtgenin boyutu üçgenin boyutunun en az iki katı olabilir. Ancak çok daha küçük bir üçgen aynı büyüklükte bir dikdörtgenle de sınırlandırabilir. Böylece üçgenin iki katından da fazla bir alan boş yere taranır. Bu nedenle Doğru Denklemi üçgen üzerinde, Yarı Alan modülünden daha az bir alanda tarama yapması ile daha avantajlı hale getirilmiştir.

$$x = \frac{(y-y2)*(x2-x1)+x2*(y2-y1)}{y2-y1} \tag{2.3}$$



Şekil 2.9: a) Üçgeni çevreleyen en küçük dörtgen (Bounding Rectangle),
b) Doğru Denklemi yöntemi ile tarama

Yukarıda verilen denklem ve mantığa göre Şekil 2.10 ve 2.11'de görüldüğü gibi yeni bir algoritma oluşturulmuştur. Bu algoritmada ilk iki bölümde bir önceki algoritmada yeralan ilk iki bölümle aynı işlemler yapılmaktadır. Bu algoritmanın üçüncü bölümünde iç içe yeralan iki döngüden bounding rectangle içinde dikey tarama yapacak olan dış döngüye girmektedir. Dördüncü bölümde her kenar için dış döngünün *y* değerlerine karşılık gelen *x* değerleri Denklem 2.3' e göre hesaplanmaktadır. Beşinci bölümde ise hesaplanan bu değerlerin yukarıda anlatılan olasılıklara göre bounding rectangle içinde olup olmadığının kontrolü yapılmaktadır. Yapılan kontrole göre belirlenen *x* değerleri arasındaki bölge algoritmanın altıncı bölümünde taranarak doldurulmaktadır. Bu algoritmada birinci algoritmadan farklı olarak üçgenin içi tek bir renk ile değil köşe noktaları için verilen renk değerlerine göre tonlama yapılarak geçerli pixelin adresine yeni renk değeri kaydedilmektedir. Bu tonlama işlemiminin yapıldığı bölüm olan altıncı bölümde, öncelikle tarama yapılacak satırın başlangıç ve bitiş pixellerinin köşe noktalarına olan uzaklığına göre olasılıklar göz önüne alınarak Denklem 2.4' e uygun şekilde renk tonlaması yapılır. Algoritmanın altıncı bölümünün devamında yer alan iç ***for*** döngüsü her satır için başlangıç ve bitiş pixelleri arasında tarama yapılırken her bir pixel için başlangıç ve bitiş pixeline olan uzaklıklarına göre renk tonlaması Denklem 2.5' e göre hesaplanarak hafızaya yazılır. Satır taraması bittiğinde ise dış döngüye döner ve Yarı Alan modülü gibi sonlanır.

```
Render(float x1, float y1, float x2, float y2, float x3, float y3)
Begin
    1. Delta değerlerini hesapla
    Dx12 = x1 - x2;  Dx23 = x2 - x3;  Dx31 = x3 - x1;
    Dy12 = y1 - y2;  Dy23 = y2 - y3;  Dy31 = y3 - y1;
    2. Sınırlayan dikdörtgeni kararlaştır
    minx = min(x1, x2, x3);  maxx = max(x1,x2, x3);
    miny = min(y1, y2, y3);  maxy = max(y1,y2, y3);
    3. Sınırlayan dikdörtgen boyunca tara
    for (y = miny; y < maxy+1; y++)
    begin
        4. Doğru denklemlerinde geçerli y noktasına karşılık gelen x
    değerlerini hesapla
        a=(((Dx12*(y1-y)) - (Dy12 * x1) ) / (-1* Dy12)) ;
        b=(((Dx23*(y2-y)) - (Dy23 * x2) ) / (-1* Dy23)) ;
        c=(((Dx31*(y3-y)) - (Dy31 * x3) ) / (-1* Dy31)) ;
        5. Doğru denklemlerinden elde edilen x değerlerini kontrol et
        if ((a>=minx and a<=maxx) and (b>=minx and b<=maxx)
                                  and (c>=minx and c<=maxx))
            if (a==b and c>a)
                minabc = a; maxabc = c+1;
            else if (a==b and c<a)
                minabc = c; maxabc = a+1;
            else if (a==c and b<c)
                minabc = b; maxabc = c+1;
            else if (a==c and b>c)
                minabc = c; maxabc = b+1;
            else if (b==c and a>b)
                minabc = b; maxabc = a+1;
            else if (b==c and a<b)
                minabc = a; maxabc = b+1;
        else if ((minx>a or a>maxx))
            if (b==c)
                minabc = b; maxabc = b+1;
            else if (b<c)
                minabc = b; maxabc = c+1;
            else if (b>c)
                minabc = c; maxabc = b+1;
        else if ((minx>b or  b>maxx))
            if (a==c)
                minabc = c; maxabc = c+1;
            else if (a<c)
                minabc = a; maxabc = c+1;
            else if (a>c)
                minabc = c; maxabc = a+1;
        else if ((minx>c or c>maxx))
                if (a==b)
                    minabc = a; maxabc = a+1;
                else if (a<b)
                    minabc = a; maxabc = b+1;
                else if (a>b)
                    minabc = b; maxabc = a+1;
        •
        •
        •
```

Şekil 2.10: Doğru Denklemi ile tarama algoritması (birinci kısım)

```
    .
    .
    .
    6. Renk yoğunluklarını hesapla
    Elde edilen satır başlangıç ve bitiş değerlerinin olasılıklar dahilinde köşe
    pixellerine olan uzaklıklarına göre renk yoğunluklarını hesapla
    switch (kontrol)
            case 1:     I4=I2;
                        I5=(((y-y3)*I1)/(y1-y3)) + ((y1-y)*I3)/(y1-y3);
                        break;
            case 2:     I4=(((y-y3)*I1)/(y1-y3))+((y1-y)*I3)/(y1-y3);
                        I5=I2;
                        break;
            case 3:     I4=(((y-y3)*I2)/(y2-y3))+((y2-y)*I3)/(y2-y3);
                        I5=I1;
                        break;
            case 4:     I4=I1;
                        I5=(((y-y3)*I2)/(y2-y3))+((y2-y)*I3)/(y2-y3);
                        break;
            case 5:     I4=I3;
                        I5=(((y-y2)*I1)/(y1-y2))+((y1-y)*I2)/(y1-y2);
                        break;
            case 6:     I4=(((y-y2)*I1)/(y1-y2))+((y1-y)*I2)/(y1-y2);
                        I5=I3;
                        break;
            case 7:     I4=I3;
                        I5=I3;
                        break;
            case 8:     I4=(((y-y3)*I2)/(y2-y3))+((y2-y)*I3)/(y2-y3);
                        I5=(((y-y3)*I1)/(y1-y3))+((y1-y)*I3)/(y1-y3);
                        break;
            case 9:     I4=(((y-y3)*I1)/(y1-y3))+((y1-y)*I3)/(y1-y3);
                        I5=(((y-y3)*I2)/(y2-y3))+((y2-y)*I3)/(y2-y3);
                        break;
            case 10:    I4=I1;
                        I5=I1;
                        break;
            case 11:    I4=(((y-y2)*I1)/(y1-y2))+((y1-y)*I2)/(y1-y2);
                        I5=(((y-y3)*I1)/(y1-y3))+((y1-y)*I3)/(y1-y3);
                        break;
            case 12:    I4=(((y-y3)*I1)/(y1-y3))+((y1-y)*I3)/(y1-y3);
                        I5=(((y-y2)*I1)/(y1-y2))+((y1-y)*I2)/(y1-y2);
                        break;
            case 13:    I4=I2;
                        I5=I2;
                        break;
            case 14:    I4=(((y-y3)*I2)/(y2-y3))+((y2-y)*I3)/(y2-y3);
                        I5=(((y-y2)*I1)/(y1-y2))+((y1-y)*I2)/(y1-y2);
                        break;
            case 15:    I4=(((y-y2)*I1)/(y1-y2))+((y1-y)*I2)/(y1-y2);
                        I5=(((y-y3)*I2)/(y2-y3))+((y2-y)*I3)/(y2-y3);
                        break;
            default:    I4=0;
                        I5=0;
    for(int x = minabc; x < maxabc; x++)
    begin
        Döngü başlangıç ve bitiş pixellerine olan uzaklığına göre geçerli pixellerin
        renk yoğunlunu hesapla
        IP=(((maxabc-x)*I4)/(maxabc-minabc))+(((x-minabc)*I5)/(maxabc-minabc));
    end for
  end for
end
```

Şekil 2.11: Doğru Denklemi ile tarama algoritması (ikinci kısım)

$$I4 = \frac{y4-y2}{y1-y2} * I1 + \frac{y1-y4}{y1-y2} * I2 \quad (2.4)$$

$$Ip = \frac{x5-xp}{x5-x4} * I4 + \frac{xp-x4}{x5-x4} * I5 \quad (2.5)$$

2.3.2. İlgili Çalışmalar

FPGA tabanlı tasarlanmış modüller yüksek performans, esneklik ve düşük maliyet gibi avantajlar sağladığı için çeşitli alanlarda kullanılmıştır. Bu alanlara örnek olarak bilgisayar teknolojisi ve bilgisayar grafikleri (Koyuncu ve Şahin, 2007), (Chodowiec ve Gaj, 2003), bulanık mantık (McKenna ve Wilamowski, 2001), görüntü ve sinyal işleme (Uzun ve diğ., 2005), (Eadie ve diğ., 2002), (Dillon, 2001), güç elektroniği (Hu ve diğ., 2006), mikroişlemcilerin geliştirilmesi (Walke ve diğ., 2000), (Borgatti ve diğ., 2002) ve özel amaçlı tasarımlar (Cavuslu ve diğ., 2006), (Beuchat, 2003), (Rouvroy ve diğ., 2003) (Visser ve diğ., 2002) şeklinde pek çok çalışmadan bahsetmek mümkündür (Koyuncu,2008).

Çalışmamızda FPGA yongaları üzerinde çalışabilecek ve rendering işlemini gerçekleştirecek donanım modülleri tasarlandığı için yapılan literatür taraması da bu yönde olmuştur. Bu yönde yapılan çalışmalardan bir tanesi Beeckler ve Gross'un (2005) yaptığı çalışmadır. Çalışma kuyu suyu, kumaş, patlamalar, yangın, duman ve bulutlar gibi modelleme olayları için uygun olan parçacık grafikleri simülasyonları üzerine hazırlanmıştır. Bu çalışmada gerçek zamanlı parçacık grafiklerini hızlandırma amaçlı FPGA üzerinde çalışabilecek bir Donanım Parçacık Makine tasarımı gerçekleştirilmiştir. Parçacık Makinesi grafik simülasyolarını ve Rendering işlemini gerçekleyen bir sistemdir. Yazılım tabanlı parçacık grafikleri simülasyonları ihtiyaç duydukları oldukça yüksek hesaplama gücü ve kaynak rekabeti sebebiyle kullanımları oldukça sınırlıdır. Çalışmanın sonucunda, FPGA yongaları kullanılarak, daha az CPU gücü gerektiren ve daha hızlı bir çözüm üretilebileceği vurgulanmıştır.

Bu alanda yapılmış çalışmalardan bir diğeri Mateusz Majer ve arkadaşlarının (2004) yapmış olduğu nokta bazlı render alanındaki çalışmadır. Çalışmada, karmaşık nesne modellerinin render edilmesine karşın ilkeller olarak anılan nokta bazlı render yaklaşımının ölçeklenebilirlik ve verimlilik açısından daha üstün olduğu kabul

edilmektedir. Sentez sonuçları ile çalışmanın karmaşık nesne modellerinin render edilmesine göre olan verimliliği ortaya koyulmuştur.

Üçgenin içinde olduğu ispatlanan pixellerin giydirilmesi işlemi de çok değişik şekillerde yapılabilmektedir. Bu alanda yapılan çalışmalardan biri ise Fender ve Rose' un (2003) ışın izleme alanındaki çalışmasıdır. Işın izleme yöntemi, sanal ışık ışınlarının 3 boyutlu sahnede ne olacağını hesaplayarak bulan yüksek kaliteli görüntü ve video işleme yöntemidir. Geleneksel Z-tamponlama yöntemlerine göre çok daha gerçekçi oluşturma yeteneğine sahiptir. Bu çalışmada bir çoklu FPGA Xilinx Virtex-E prototipleme sisteminde geliştirilen donanım ışın izleme sisteminin tasarımı anlatılmaktadır. Işın izleyici donanımı iyi bilinen bir yüksek performanslı ışın izleme algoritması yazılımına oranla otuz kat daha verimlidir.

FPGA üzerine bu çalışmalar dışında birçok farklı alanda yapılan çalışma da mevcuttur. Bu çalışmalardan biri de Sırmaçek'in (2007) yapmış olduğu mobil robot için öğrenme algoritması modellenmesidir. Sırmaçek çalışmasında mobil bir robot uygulaması için çevreyi ve engellerden kaçmayı öğrenecek bir YSA algoritması gelişirmiş ve bu algoritmanın FPGA için donanımsal modellemesini yapmıştır. Algoritma olasıksal sinir ağı (Probabilistic Neural Network (PNN)) yapısı kullanılarak modellenmiş ve PNN'e bu çalışmadaki dezavantajlarını düzeltecek bazı üstünlükler kazandırılmıştır. Delphi ortamında engellerin, ulaşılması istenen hedef noktanın istenilen sekilde yerleştirilmesine musaade eden bir bilgisayar grafik arayüzü hazırlanmıştır. Bu arayüz kullanıcıya mobil bir robotun engellerin olduğu bir odada hedef noktaya en kısa yoldan varmasını ve engel koordinatlarını öğrenmesini göstermektedir. Mobil robot, başla sinyalini aldığında yavaş yavaş hedef noktasına doğru en kısa yolu bularak o yolda ilerlemekte ve bu sırada önüne çıkan engellerden kaçmaktadır. Engelleri algılama, doğrultusundan ne kadar açı ile sapacağına karar verme ve tekrar hedefe konumlanma için oluşturulan yeni bir algoritmadan yararlanılmıştır. Arayüz simulasyonu engellerin bulunduğu noktaları öğrenerek koordinatlarını kullanıcıya ulaştırmakta, böylece robot odayı da öğrenmiş olmaktadır.

FPGA üzerine yapılan çalışmalardan bir diğeri Yılmaz'ın (2008) bir YSA' nın donanımsal olarak tasarlanması ve gerçekleştirilmesi alanında yapmış olduğu çalışmadır. Yılmaz

çalışmasında, Altera FPGA devreleri ile eğitilebilir bir YSA yapısı gerçekleştirmiştir. Çalışmada XOR problemi ve bir sensör doğrusallaştırma problemi ile çalışılmış ve sabit noktalı sayı sistemi tabanlı ve hatanın geri yayılımı algoritması ile eğitilen bir YSA yapısı kullanılmıştır. Delta bar delta öğrenme kuralı kullanılmıştır. Altera'nın QUARTUS II FPGA tasarım programı ve MATLAB ile uygulamalar tasarlanmış ve simülasyonu gerçekleştirilmiştir. Ayrıca, basitleştirilmiş YSA yapısı ile XOR problemi, Altera Cyclone EP1C6Q240C8 FPGA tabanlı UP3 geliştirme kartı kullanılarak gerçeklenmiştir. Yapılan bu çalışma sayesinde bazı YSA tabanlı sistemler için FPGA'nın maliyet, zaman tasarrufu, yeniden yapılandırılabilirlik ve paralel tasarım yeteneği yönlerinden daha avantajlı bir çözüm olduğu gösterilmiştir.

3. MATERYAL VE YÖNTEM

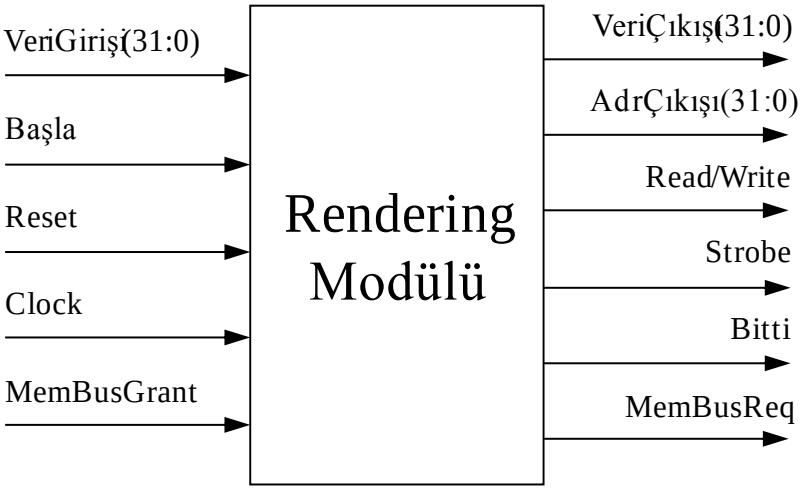
3.1. BİLGİSAYAR GRAFİKLERİ İÇİN FPGA TABANLI ÜÇGEN DOLDURMA MODÜL TASARIMI

Yüzey giydirme işlemlerinde pek çok yöntem kullanılmaktadır. Ancak Çoğunlukla yazılım tabanlı kullanılan algoritmalardan istenen verim alınamamaktadır. Bu çalışmada bilgisayar grafiklerinde rendering işlemini hızlandırmak için FPGA yongaları üzerinde çalışabilecek iki farklı donanım modülü tasarlanmıştır. Modüller, çalışmanın ikinci bölümünde anlatılan doğru denkleminden yararlanılarak oluşturulan algoritmalar temel alınarak tasarlanmıştır. İlk modül, yarı alan formülünü kullanarak tarama yapılan alan içindeki geçerli pixelin üçgenin içinde olup olmadığına karar veren Yarı Alan modülüdür. Bu modülde üçgeni çevreleyen en küçük dörtgen bulunup, sırayla taranak geçerli pixelin kontrolü yapılarak verilen sabit renge boyanmaktadır. İkinci modül ise Doğru Denkleminden yararlanılarak oluşturulan Doğru Denklemi modülüdür. Bu modülde ilk tasarımdan farklı olarak üçgeni çevreleyen dörtgen üzerinde değil sadece üçgen içinde tarama yapılmakta ve buna ek olarak boyanacak pixeller sabit bir renge değil her bir pixele ait yoğunluk hesaplanarak o renge boyanmaktadır.

Modüller öncelikle Register Transfer Logic (RTL) seviyesinde kağıt üzerinde tasarlanmış ardından bir donanım tasarım dili olan Çok Yüksek Hızlı Tümleşik Devre Tanımlama Dilinde (VHDL) kodlanmıştır. İlk modül XILINX ISE 9.2, ikinci modül ise XILINX ISE 12.1 Elektronik Tasarım Otomasyon aracı (EDA) kullanılarak sentezlenmiştir. Bu bölümde öncelikle her iki modülün ortak yapısı ardından her bir modülün detayları anlatılacaktır.

3.1.1. Modüllerin Genel Yapısı

Şekil 3.1' de her iki modülün ortak en üst seviye blok diyagramı görülmektedir. Modüllerin adres ve veri yolları 32 bit olarak tanımlanmıştır. Bu sayede 4 GB'lık bir hafıza alanı adreslenebilmekte ve 32 bitlik standart veriler işlenebilmektedir.

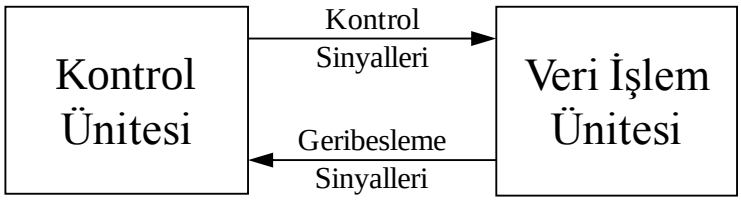


Şekil 3.1: Rendering modüllerinin ortak en üst seviye blok diyagramı

Modüller ile ana bilgisayar aradındaki koordinasyonu (handshaking) sağlamak için `Reset`, `Basla` ve `Bitti` sinyalleri kullanılmıştır. `Reset` sinyali ile modüller bekleme durumuna getirilmekte, `Basla` sinyali ile ise modüllerin verilen bir üçgen alanını render etmeye başlaması sağlanmaktadır. Modüller bir üçgeni render ettikten sonra hafızadan diğer bir üçgenin bilgilerini okuyarak işleme devam eder. Bütün üçgenler bittikten sonra ana bilgisayara `Bitti` sinyalini göndererek yeni bir işlem için beklemeye başlamaktadırlar. Modüllerin standart bir SDRAM hafıza kullandıkları kabul edilerek bu hafızaya erişmek için ayrıca `MemBusReq` (veri yolu talep sinyali), `MemBusGrant` (veri yolunun module tahsis edildiğini belirten sinyal), `Strobe` (hafızada işlem yapıldığını belirten sinyal) ve `Read/Write` (hafızada okuma yada yazma işelemini belirten sinyal) tanımlamıştır. Bu sinyaller sayesinde modüller hafızadan render edilecek üçgene ait bilgileri okumakta, yine aynı hafızada bulunan ekran hafızasında üçgenin içinde olduğuna karar verilen geçerli pixelin belirtilen adresine renk bilgisini yazmaktadırlar.

Şekil 3.2'de modüllerin ortak İkinci Seviye blok diyagramı verilmiştir. Modüller, *Kontrol Ünitesi* ve *Veri İşlem Ünitesi* olmak üzere iki bölüm olarak tasarlanmışlardır.

Kontrol Ünitesi, uygun zamanlarda gerekli kontrol sinyallerini üreterek, hafızadan verilerin alınmasını, alınan bu verilerin *Veri İşlem Ünitesi* üzerinde işlenmesini ve elde edilen sonuçların hafızaya yazılmasını sağlar. *Veri İşlem Ünitesi* ise *Kontrol Ünitesinden* kullanılan algoritmaya uygun olarak gelen kontrol sinyalleri ile verileri uygun şekilde işleyerek sonuçları üretir.



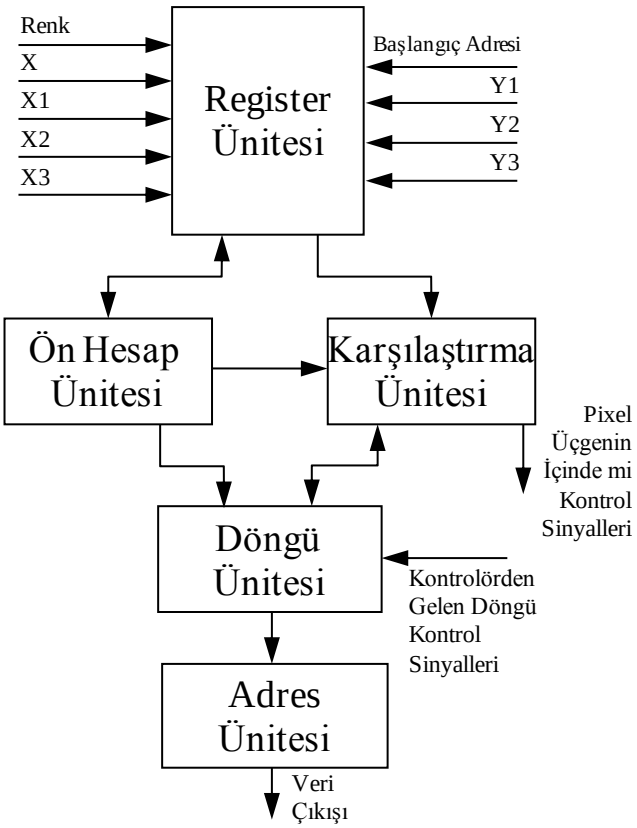
Şekil 3.2: Modüllerin ikinci seviye blok diyagramı

Kullanılan algoritmaya bağlı olarak *Kontrol Ünitesinin* ve *Veri İşlem Ünitesinin* yapısı değişmektedir. Takip eden bölümlerde her bir algoritma için geliştirilen *Kontrol* ve *Verişleme Ünitelerinin* yapıları detaylı olarak anlatılacaktır.

3.1.2. Yarı Alan Modülü

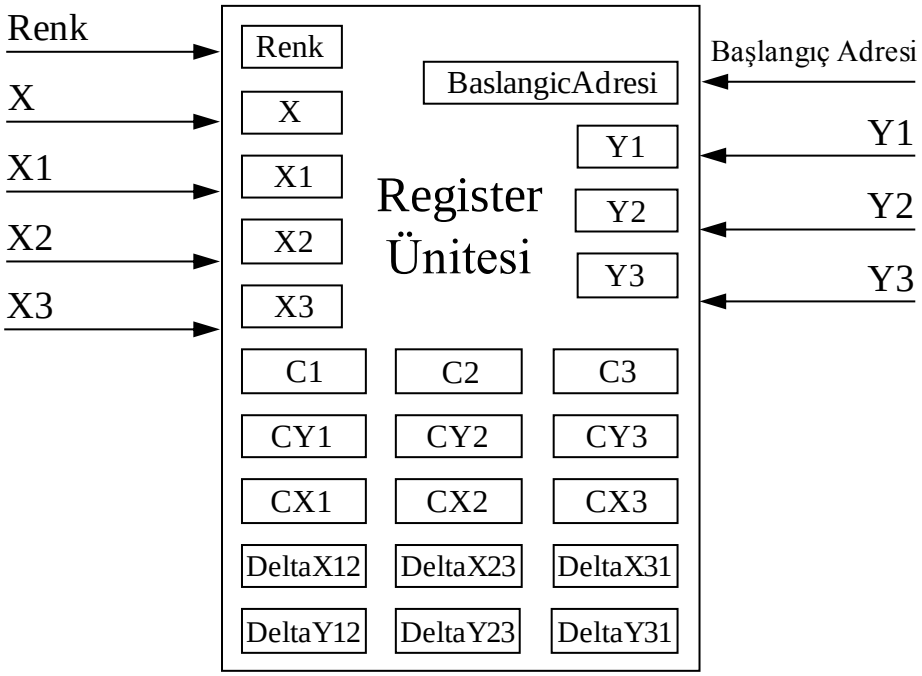
Şekil 3.3'de Yarı Alan modülü *Veri İşlem Ünitesinin* blok diyagramı görülmektedir. Bu ünitede dört ana blok yer almaktadır.

Register Ünitesi, hafızadan okunan üçgen koordinatları, ekran hafızası başlangıç adresi ve bu verilerle yapılan hesaplamalardan elde edilen ara değerlerin kaydedildiği registerların bulunduğu ünitedir. *Ön Hesaplama Ünitesinde*, kullanılan rendering algoritmasına bağlı olarak, yarı alan formulü için gerekli ön hesaplamaları yapmaktadır. Hesaplanan bu değerler yine *Register Ünitesi* içinde registerlarda saklanmakta ve üçgenin bulunduğu bölgeyi tararken kullanılmaktadır. *Döngü Ünitesi* algoritmada yer alan iç içe ***for*** döngülerinin gerçekleştirilmesinden ve bu döngüler içinde yapılan hesaplamalardan sorumludur. *Karşılaştırma Ünitesi* ise üçgeni çevreleyen bounding rectangle alanın içindeki geçerli koordinatdaki pixelin, üçgenin içinde olup olmadığının belirlenmesinde kullanılmaktadır. Eğer herhangi bir pixelin üçgenin içinde veya üzerinde olduğu tespit edilirse, bu durum kontrolöre bildirilerek geçerli pixelin bulunduğu hafıza adresi hesaplanarak veri çıkışından önce bulunan *Adres Ünitesinde* yeralan *AdresReg Registerına* adres bilgisi yazılarak bu adrese *Renk Registerına* kaydedilen renk bilgisinin yazılması sağlanır.



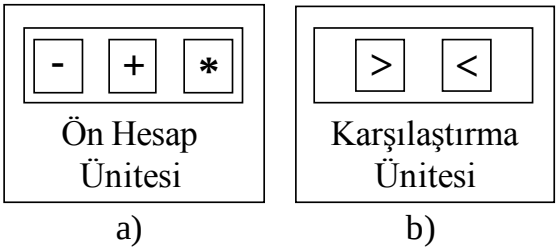
Şekil 3.3: *Veri İşlem Ünitesi* blok diyagramı

Şekil 3.4'te *Register Ünitesi'*nin blok diyagramı ayrıntılı olarak görülmektedir. Bu ünitede üçgen koordinat verilerinin okunacağı hafıza adresnini tutan *BaslangıçAdresi Registeri*, üçgen koordinat değerlernini saklandığı *X1, X2, X3, Y1, Y2, Y3 Registerları*, yarı alan hesaplamaları için gerekli değerlerin tutulduğu *C1, C2, C3, CY1, CY2, CY3, DX12, DX23, DX31, DY12, DY23, DY31 Registerları*, üçgenin içinin doldurulacağı renk bilgisinin tutulduğu *Renk Registerı* ve çizimin yapıldığı grafik alanın genişlik bilgisinin tutulduğu *X Registerları* bulunmaktadır.

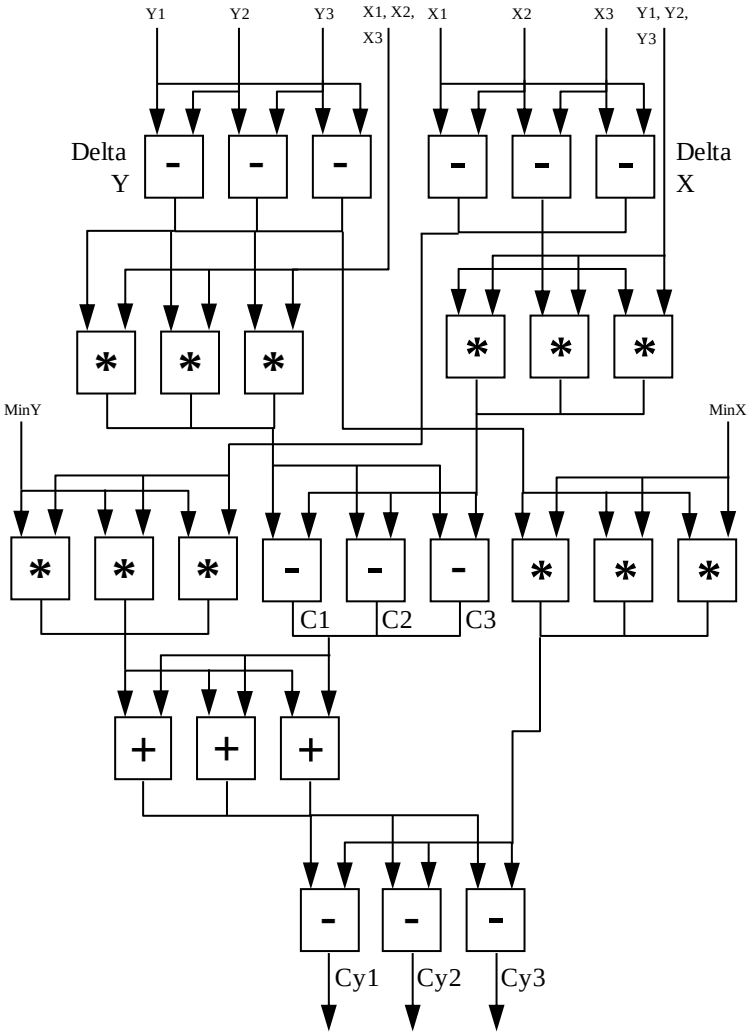


Şekil 3.4: *Register Ünitesi* blok diyagramı

Şekil 3.5 a'da *Ön Hesap Ünitesi*, b'de ise *Karşılaştırma Ünitesi'*nin blok diyagramları görülmektedir. Şekil 3.6 'da blok diyagramı görülen *Ön Hesap Ünitesi*, Şekil 3.7'de görülen algoritma parçasında yer alan yarı alan hesaplamalarını gerçekleştirir. Algoritma gereği hesaplanan *C1, C2, C3, CY1, CY2, CY3, DX12, DX23, DX31, DY12, DY23, DY31* değerleri hesaplama işleminden sonra *Register Ünitesindeki* ilgili registerlara kaydedilir. Şekil 3.8' de blok diyagramı görülen *Karşılaştırma Ünitesi* ise algoritmada yeralan en küçük ve en büyük *x,y* değerlerini seçmek ve ikinci bölümde verilen Şekil 2.7' deki algoritmada yer alan iç döngü altında işleyen ***if*** komutuna ait karşılaştırmaları yapmak üzere tasarlanmıştır. Bu iki ünite içindeki hesaplamalar ve karşılaştırma işlemleri paralel olarak gerçekleştirilmekte ve zamandan kazanç sağlanmaktadır.



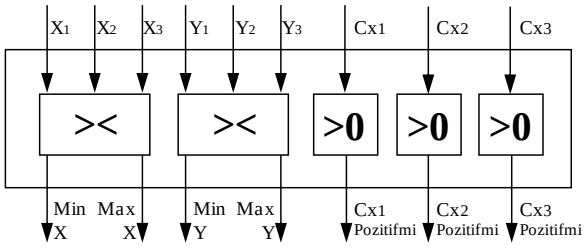
Şekil 3.5: a) *Ön Hesap Ünitesi* blok diyagramı b) *Karşılaştırma Ünitesi* blok diyagramı



Şekil 3.6: *Ön Hesap Ünitesi* blok diyagramı

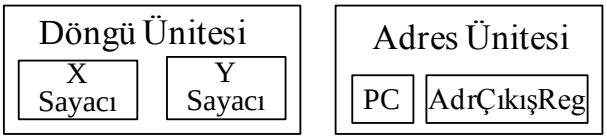
```
•
•
•
Delta değerlerini hesapla
Dx12 = x1 - x2;  Dx23 = x2 - x3;  Dx31 = x3 - x1;
Dy12 = y1 - y2;  Dy23 = y2 - y3;  Dy31 = y3 - y1;

Yarı kenar fonksiyonunun sabit parçalarını hesapla
C1 = Dy12 * x1 - Dx12 * y1;
C2 = Dy23 * x2 - Dx23 * y2;
C3 = Dy31 * x3 - Dx31 * y3;
Cy1 = C1 + Dx12 * miny - Dy12 * minx;
Cy2 = C2 + Dx23 * miny - Dy23 * minx;
Cy3 = C3 + Dx31 * miny - Dy31 * minx;
•
•
•
```

Şekil 3.7: Yarı Alan algoritması için gerekli ön hesaplamaları gerçekleştiren algoritma parçası



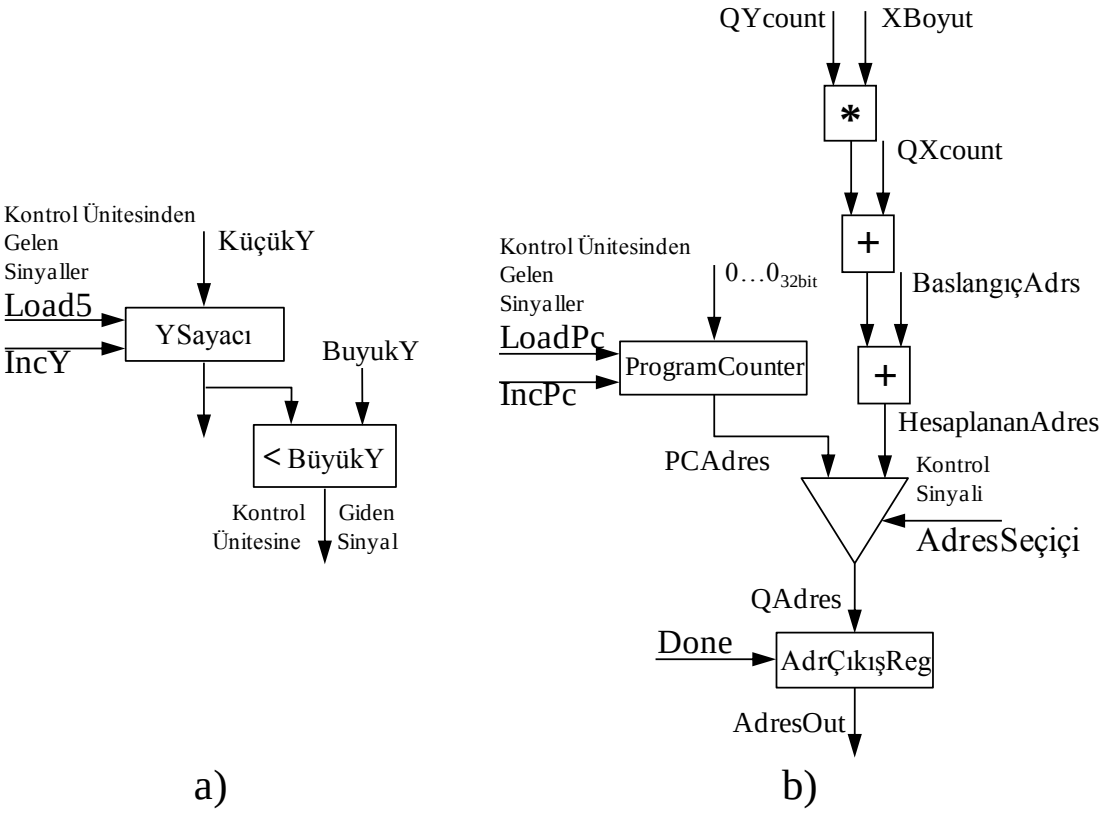
Şekil 3.8: *Karşılaştırma Ünitesi* blok diyagramı

Şekil 3.7 a'da *Döngü Ünitesi* ve b' de *Adres Ünitesine* ait blok diyagramları görülmektedir. *Döngü Ünitesi*, ikinci bölümde verilmiş olan Şekil 2.7'deki Yarı Alan algoritmasının çalışmasını sağlayan içi içe iki ***for*** döngüsünü gerçekleştirmektedir. Algoritmada yer alan döngüler *Döngü Ünitesinde YSayacı* ve *XSayacı* olarak tanımlanmıştır. Şekil 3.10 a' da örnek olarak *YSayacı* blok diyagramı görülmektedir. Bu döngüler sayesinde render edilecek üçgeni çevreleyen dikdörtgen alan iki boyutlu olarak taranmakta ve bu dikdörtgen alan içindeki her bir pixelin üçgenin içinde olup olmadığı test edilmektedir. Dikdörtgen içindeki herhangi bir pixelin üçgen içinde olup olmadığı Şekil 3.5 b'de görülen *Karşılaştırma Ünitesi* tarafından belirlenmektedir. Eğer geçerli nokta üçgen içerisinde ise *Karşılaştırma Ünitesi* bunu bir kontrol sinyali ile *Kontrol Ünitesine* bildirir. *Kontrol Ünitesi* ise pixelin hafıza adresinin hesaplanması ve bu adrese renk bilgisinin yazılması için gerekli olan `AdresSecici` ve `Done` sinyalleri üretir. `AdresSecici` Şekil 3.10 b' de blok diyagramı görülen *Adres Ünitesinde ProgramCounter* sayacı çıkışından elde edilen PCAdres ya da hesaplanarak elde edilen adres bilgisini ifade eden HesaplananAdres değerlerinden hangisinin seçileceğine karar verir. Burada *ProgramCounter* modül ilk çalışmaya başladığında hafızaya erişimi sağlamak için hafıza erişim adres bilgisini üreten sayaçtır. Hesaplanan adres ise üçgen içinde olduğu tespit edilen pixelin hesaplanarak elde edilen ekran hafızasındaki adresidir. `Done` sinyali ise multiplexer çıkışına iletilen adres bilgisinin *AdresReg Registerına* kaydedilmesini kontrol ederek hafıza adresini kaydeder.



a) b)

Şekil 3.9: a) *Döngü Ünitesi* blok diyagramı, b) *Adres Ünitesi* blok diyagramı



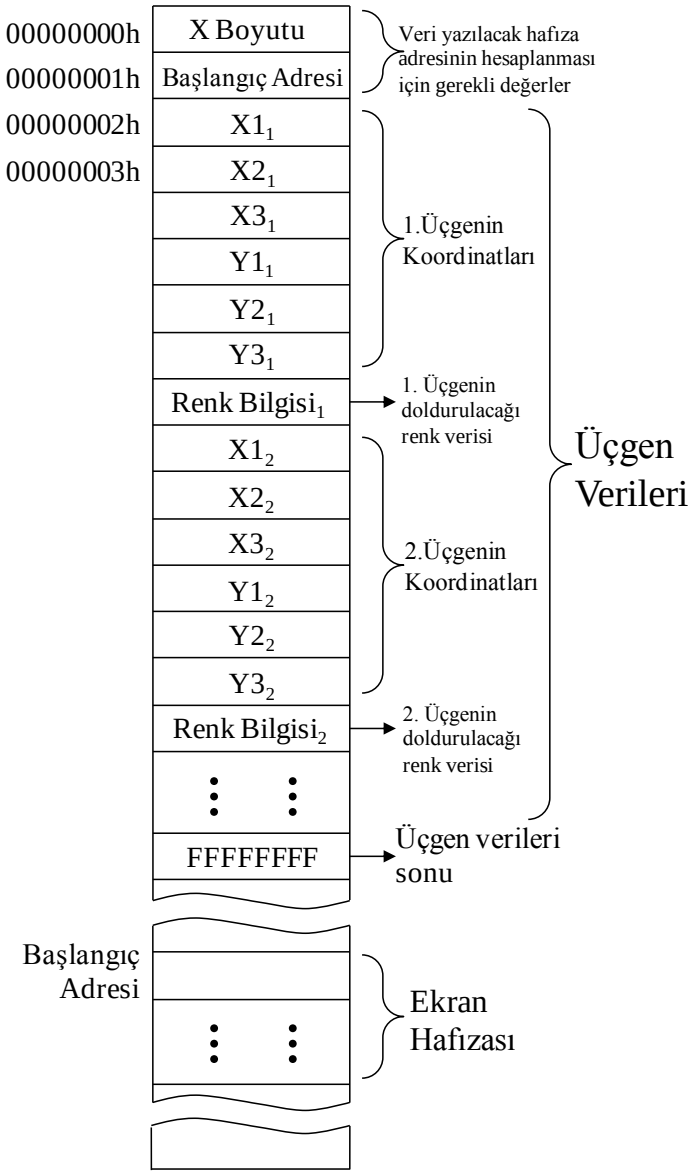
Şekil 3.10: a) *YSayacı* blok diyagramı, b) *Adres Ünitesi* blok diyagramı

3.1.2.1. Yarı Alan Modülü Hafıza Haritası ve Kontrol Ünitesi

Şekil 3.11'de tasarlanan *Yarı Alan Modülünün* kullandığı hafıza haritası görülmektedir. Adreslenebilir herbir veri alanının 32-bit olduğu ve modüller için gerekli verilerin 00000000_h adresinden itibaren sıralı olarak yeraldığı varsayılmıştır. Bu alanlara erişim ise *Program Counter* sayacı kullanılarak gerçekleştirilmiştir.

Hafıza haritasında üç bölüm yer almaktadır. Bu bölümler; verilerin yazılacağı hafıza adresinin hesaplanması için gerekli ekran hafızası bilgileri, üçgen verileri ve yeni verilerin yazılacağı ekran hafıza alanından oluşmaktadır.

İlk iki hafıza adresinde kontrolü yapılan pixel için hesaplanan renk bilgisinin yazılacağı adresin hesaplanmasında kullanılacak olan ekranın *X* genişliği ve Başlangıç Adresi yer almaktadır. $0…2_h$ adresinden itibaren tasarlanan modülün işleyeceği üçgen verileri yer almaktadır. Bu veriler üçgen koordinatları ve renk bilgisinden oluşmaktadır. Bu alanın bitişi $FFFFFFFF_h$ bilgisi ile işaretlenmiştir.



Şekil 3.11: Yarı Alan Modülü hafıza haritası

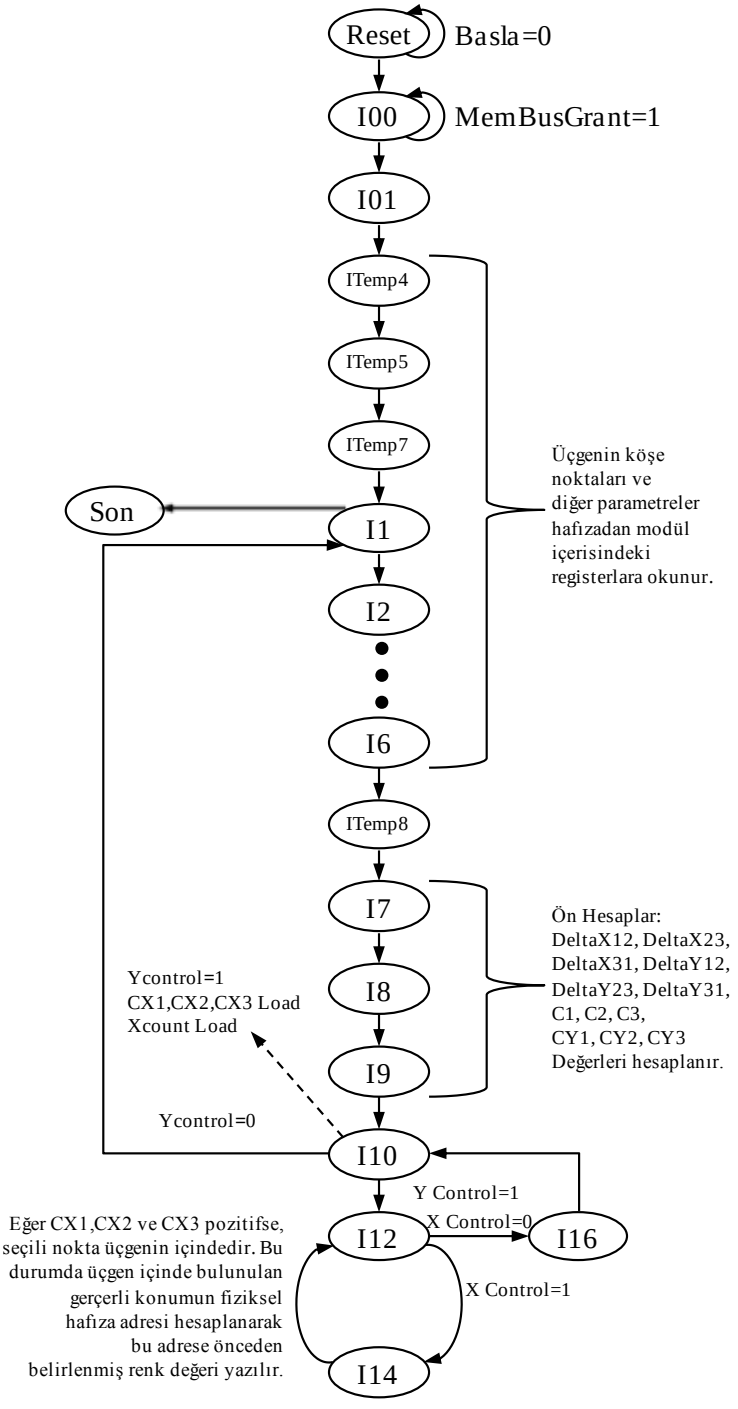
Şekil 3.12'de modül kontrolürünün çalışmasını gösteren *durum diyagramı* görülmektedir. Kontrolör toplam 20 *durum*'dan oluşan bir Sonlu Durum Makinesi (Finite State Machine(FSM)) olarak tasarlanmıştır. Modül ilk çalışmaya başladığında kontrolör *Reset durum*'unda host bilgisayardan `Basla` sinyali gelene kadar bekler. Bilgisayardan `Basla` sinyalini aldığında, *Reset durum*'undan çıkarak, hafızaya erişmek için sistem yoluna istek sinyali (`MemBusReq`) gönderir ve yolun kendine tahsis edilmesini bekler. Yol tahsis edildikten sonra (`MemBusGrant` sinyalini alınca), kontrolör hafızada $0\ldots0_h$ adresine giderek render edilecek üçgenin grafik alanının hafıza

bilgilerini (ekran hafıza alanın başlangıç adresi ve çizim alanının genişliği) okur. Bu bilgiler üçgen içindeki render edilecek pixellerin fiziksel hafıza adresinin hesaplanmasında kullanılır. Ardından kontrolör takip eden hafıza adreslerinden üçgenin köşe noktalarının koordinatlarını ve üçgenin doldurulacağı renk bilgisini okur. Bu işlemler toplam 12 *durum* boyunca yapılır ve okunan bu veriler *Register Ünitesinde* ilgili registerlara kaydedilir. *I7 durumunda* tarama işlemi sırasında geçerli noktanın konumunun kontrolünü yapmak için kullanılacak olan *deltax12, deltax23, deltax31, deltay12, deltay23, deltay31* fark değerleri; *I8 durumunda c1, c2, c3* değerleri; *I9 durumunda* ise *cy1, cy2, cy3* değerleri hesaplanarak üçgeni çevreleyen en küçük dikdörtgenin koordinatları belirlenir. Bu aşamadan sonra kontrolör algoritmada yer alan iç içe döngüleri işletmeye başlar. İç döngü *I12* ve *I14* durumlarında gerçekleştirilir. Dış döngü ise *I10,* iç döngü durumları (*I12*, *I14*) ve *I16* olmak üzere toplam dört *durumda* gerçekleştirilir. Dış döngünün her tekrarlanışında üçgeni çevreleyen dikdörtgenin bir satırının taraması yapılır. İç döngünün her tekrarlanışında ise taranan satırdaki bir pixelin durumu (üçgenin içinde olup olmadığı) belirlenir, eğer geçerli pixel üçgen içinde ise bu pixelin fiziksel hafıza adresi hesaplanarak bu adrese önceden belirlenmiş renk bilgisinin yazılması sağlanır. Döngüler tamalanlandığında verilen üçgenin render edilme işlemide tamamlanmış olur. Kontrolör bu aşamadan sonra yeni bir üçgenin koordinat değerlerini ve doldurulacağı renk bilgisini hafızadan okumak üzere *I1 durumuna* geri döner. Bu işlem hafızada verilen bütün üçgenler render edilene kadar devam eder. Kontrolör render edilecek üçgenlerin bittiğini *I1 durumunda* hafızadan okuduğu değere bakarak karar verir. Bu aşamada okunan değer eğer $\texttt{FFFFFFFF}_{\texttt{h}}$ ise bunun anlamı verilen bütün üçgenler render edilmiştir. Dolayısıyla kontrollor *Stop durumuna* giderek yeni bir veri setini işlemek üzere resetlenmeyi beklemeye başlar. Kontrolör bu şekilde beklerken host bilgisayar hafızaya yeni üçgen verilerini yazarak hafızayı tazeler ve modülü tekrar başlatır.

Kontrolör her bir pixelin durmunu belirlemek için 2 saat darbesi harcar. Genişliği x ve yüksekliği y olan bir dikdörtgenin bir satırının taranması için toplam $2x+2$ saat darbesi harcanır. Dikdörgenin tamamının taranması için ise toplam $(2x + 2) * y$ adet saat derbesi harcanır. Bir üçgene ait bilgilerin hafızadan okuması için 12 saat darbesine ihtiyaç duyulur. Bu değer de formule dâhil edildiğinde dikdörtgen için harcanan saat darbesi adedi;

$$\left((2x+2)*y+12\right) \tag{3.1}$$

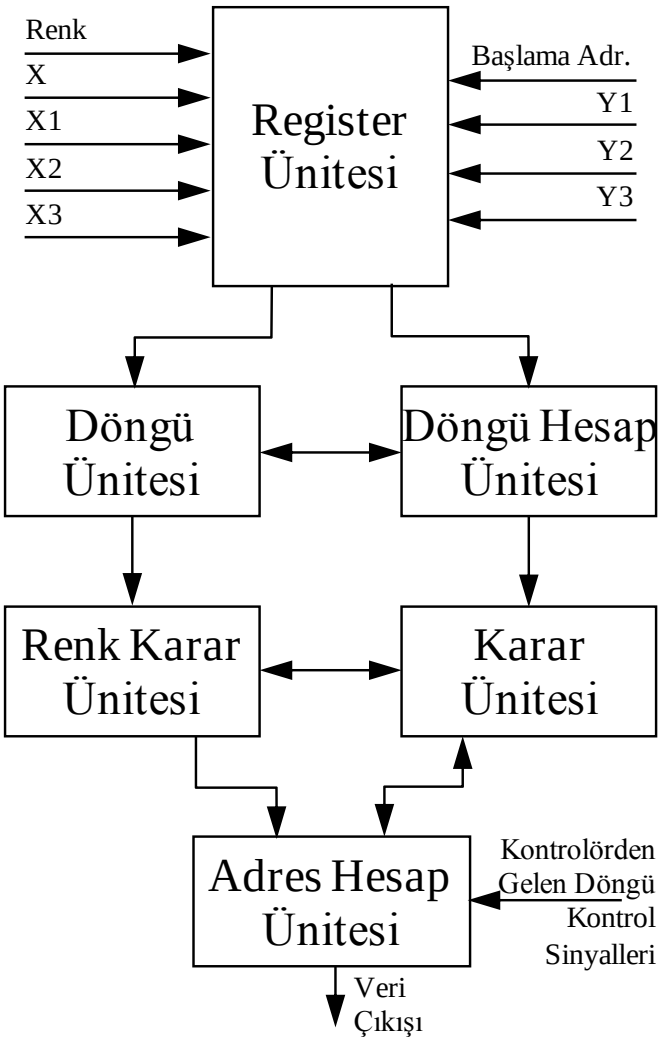
formülü ile hesaplanabilir.



Şekil 3.12: Modül *Kontrol Ünitesi Durum* diyagramı

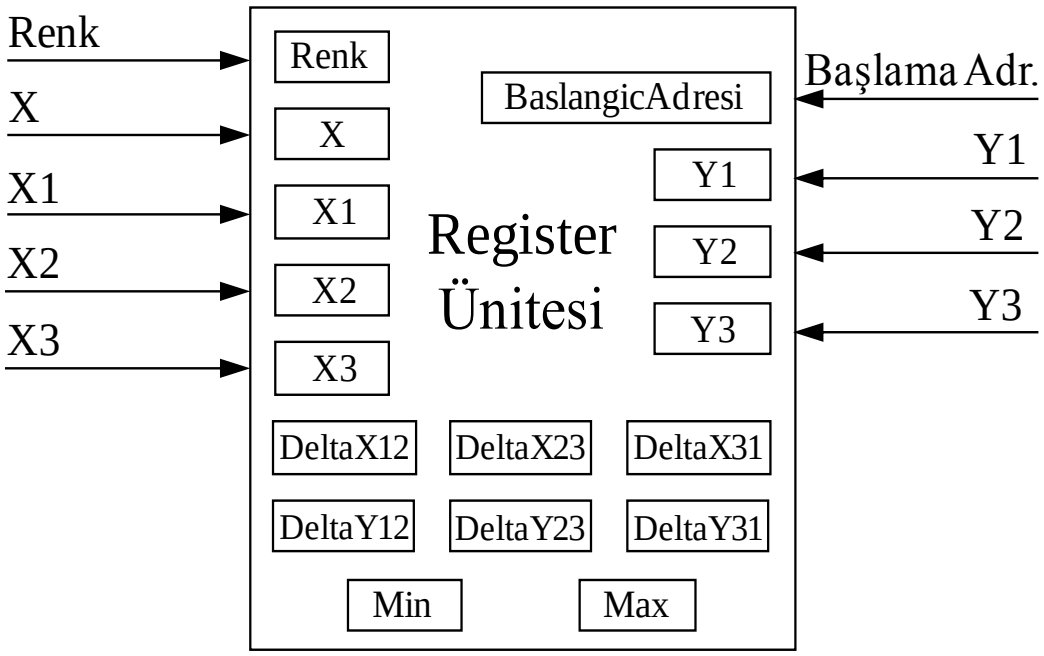
3.1.3. Doğru Denklemi Modülü

İlk tasarlanan modül algoritmasında çeşitli optimizasyonlar yapılarak yeni bir algoritma ve bu yeni algoritmaya bağlı olarak yeni bir modül (Doğru Denklemi Modülü) tasarlanmıştır. Doğru denklemi modülünün en üst seviye ve ikinci seviye blok diyagramları birinci modül ile aynı olup Şekil 3.1 ve 3.2'de görüldüğü gibidir. Modülün dış yapısında herhangi bir değişiklik yapılmamış *Veri İşlem Ünitesi* ve *Kontrol Ünitesi* yeni algoritmaya uygun olarak yeniden tasarlanmıştır. Şekil 3.13' te Doğru Denklemi Modülü için tasarlanan ve altı ana bloktan oluşan *Veri İşlem Ünitesinin* blok diyagramı görülmektedir.



Şekil 3.13: *Veri İşlem Ünitesi* blok diyagramı

Şekil 3.14'te *Register Ünitesi'*nin blok diyagramı ayrıntılı olarak görülmektedir. *Register Ünitesi*'nde işlenmiş üçgen verilerinin yazılacağı hafıza adresini tutan *BaslangicAdresi Registerı*, üçgen koordinat değerlerinin saklandığı *X1, X2, X3, Y1, Y2, Y3 Registerları,* doğru denklemindeki hesaplamalar için gerekli delta değerlerinin tutulduğu *DX12, DX23, DX31, DY12, DY23, DY31 Registerları*, üçgenin içinin doldurulacağı renk bilgisinin tutulduğu *Renk Registerı*, çizimin yapıldığı grafik alanın genişlik bilgisinin tutulduğu *X Registerı* ve *Döngü Hesap Ünitesi'*nde hesaplanan değerlerin tutulduğu *Min* ve *Max Registerları* bulunmaktadır. Yarı Alan Modülünde yer alan *Döngü Ünitesi* ve *Adres Ünitesi* bu modülde de değişiklik yapılmadan kullanılmıştır.



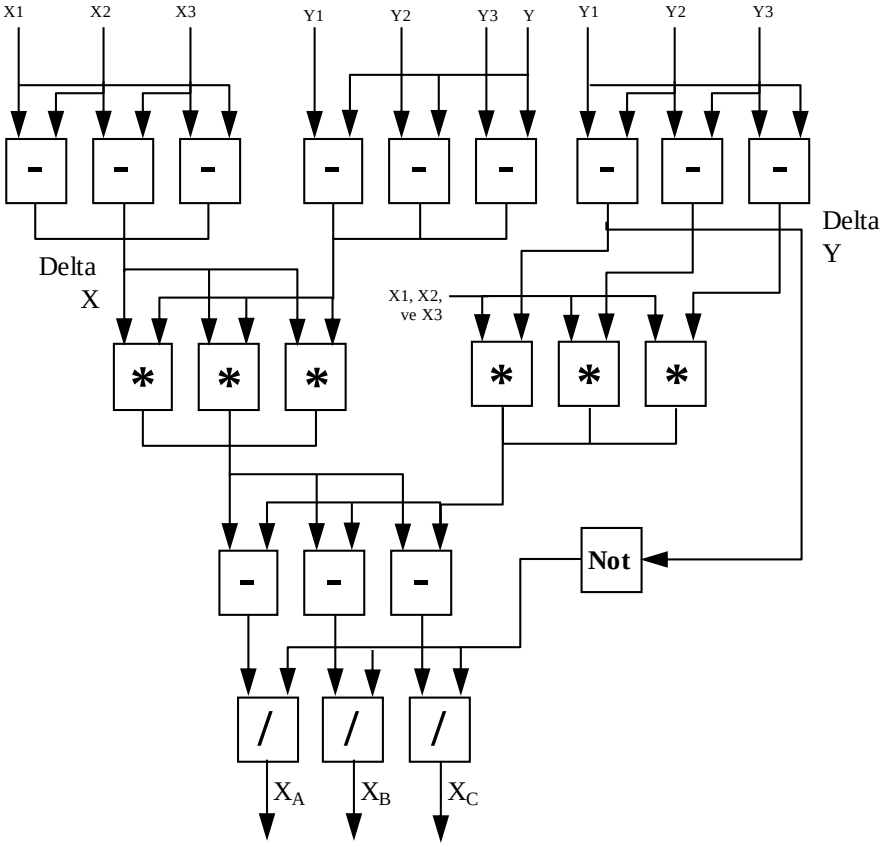
Şekil 3.14: *Register Ünitesi* blok diyagramı

Sadece üçgen üzerinde tarama yapılabilmesi için her doğru denkleminde geçerli *y* değerine karşılık gelen *x* değerlerini Şekil 3.15'teki algoritma parçasında görüldüğü gibi hesaplayan *Döngü Hesap Ünitesi* blok diyagramı Şekil 3.16'da görülmektedir. Bu ünitede X_A, X_B ve X_C çıkışları doğru denklemi formüllerinde geçerli *y* değerine karşılık gelen *x* değerlerini temsil etmektedir.

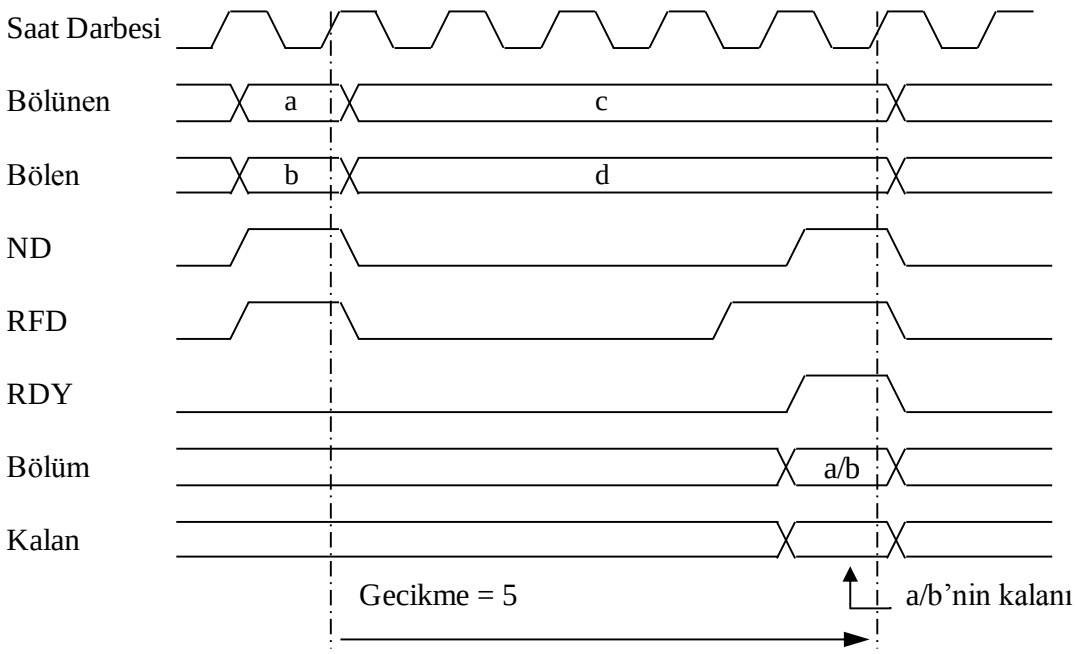
```
•
•
•
Doğru denklemlerinde geçerli y noktasına karşılık gelen x değerlerini hesapla
a=(((Dx12*(y1-y)) - (Dy12 * x1) ) / (-1* Dy12)) ;
b=(((Dx23*(y2-y)) - (Dy23 * x2) ) / (-1* Dy23)) ;
c=(((Dx31*(y3-y)) - (Dy31 * x3) ) / (-1* Dy31)) ;
•
•
•
```

Şekil 3.15: Doğru denklemlerinde her doğru parçasında geçerli *y* değerine karşılık gelen *x* değerlerinin hesaplamalarını gerçekleştiren algoritma parçası

Yukarıda görülen işlemler için gerekli olan *Çıkarma, Çarpma* ve *Bölme* üniteleri için tasarım yapılmamış, bu üniteler direk olarak VHDL'e kodlanmış ve sentez aracı tararafından otomatik olarak oluşturulmuştur. *Çıkarma* ve *Çarpma* üniteleri paralel olarak çalışmaktadır ve kombinasyonel matık devresi olarak tasarlandıklarından, girişlerine veri ulaştığı andan kısa bir süre sonra (nano saniyeler içinde) sonucu üretmektedirler. *Bölme Ünitesi* sentez aracı ile direk olarak oluşturulamadığından, yine Xilinx firması tarafından geliştirilen ve ISE'nin bir parçası olan IP Core Generator aracı kullanılarak oluşturulmuştur. *Bölme Ünitesi* Şekil 3.17'de görüldüğü gibi `ND` (New Data (Yeni Veri)), `RFD` (Ready For Data (Veri İçin Hazır)) ve `RDY` (Ready (Hazır)) sinyalleri ile kontrol edilmektedir. Sinyallerin hepsi yükselen kenarda aktiftir. Bölme Ünitesinin bölme işlemi yapmak üzere veri almaya hazır olduğunda `RFD` sinyali üreterek kotrolörü haberdar eder. Ünitenin girişinde data hazır olduğunda `ND` sinyalinin kontrolör tarafından aktif yapılması gerekmektedir. Ünite bölme işlemini bitirdiğinde sonucun hazır olduğunu belirtmek için `RDY` sinyalini aktif yapar. Bu üç sinyal sayesinde Kontrollör ile bölme ünitesi arasındaki koordinasyon gerçeleştirilir. Bölme ünitesinin gecikmesi (Latency) altı saat darbesidir. Dolayısıyla *Bölme Ünitesi* girişindeki verileri okuyarak işlemeye başladıktan altı saat darbesi sonunda sonucu *Bölüm* çıkışında hazır hale getirir.

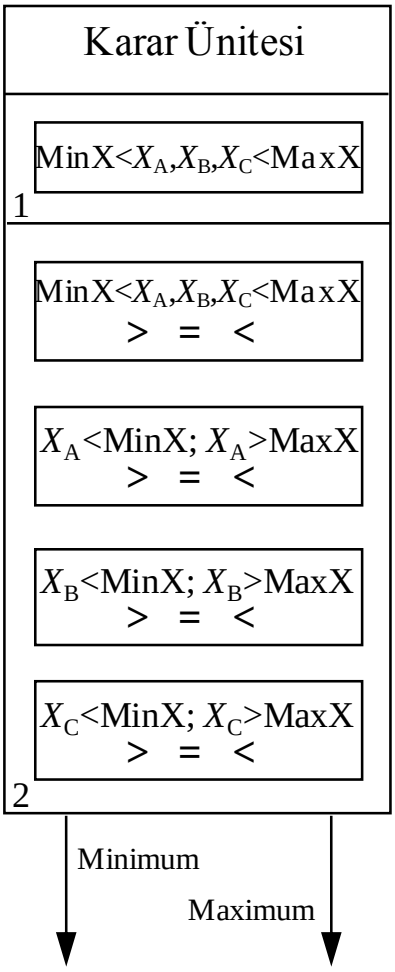


Şekil 3.16: *Döngü Hesap Ünitesi* blok diyagramı

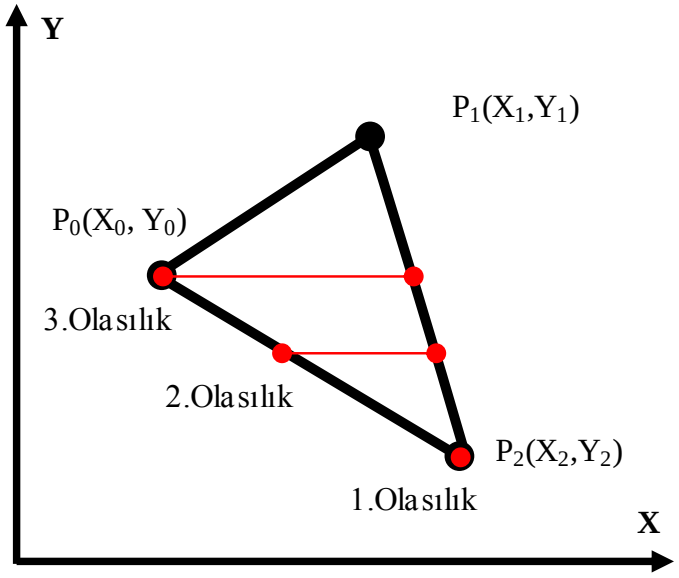


Şekil 3.17: *Bölme Ünitesi* zaman diyagramı (Anon, 10 Ağustos 2011)

Şekil 3.18'de *Karar Ünitesi* sembolik blok diyagramı görülmektedir. Şekilde görülen birinci kısımda *Döngü Hesap Ünitesinde* elde edilen X_A, X_B ve X_C değerlerinin *if* blokları ile *MinimumX* ve *MaximumX* değerleri arasında olup olmadığının kontrolü yapılır. İkinci kısımda ise ilk şart sağlandığı takdirde, ilk şarttan elde edilecek sonucada bağlı olarak, X_A, X_B, X_C değerlerinden hangisinin Minimum hangisinin Maximum çıkışına iletileceği kontrolü yapılmaktadır. Ayrıca ikinci kısım kontrolü yapılırken Şekil 3.19' da görülen olasılıklar göz önüne alınmaktadır. Bu kontroller Şekil 3.20'de verilen doğru denklemi program parçasından yararlanılarak oluşturulmuştur. Yapılan kontrollerde üçgenin şekil ve konumuna bağlı olarak özellikle Şekil 3.19' da görülen olasılıklar üzerinde durulmuştur. Bunun sebebi üçgenin şekil ve konumu gereği koordinatların düzgün, yani hesaplamalarda herhangi bir hataya sebep olmayacak şekilde, olmaması ihtimalidir.



Şekil 3.18: *Karar Ünitesi* blok diyagramı



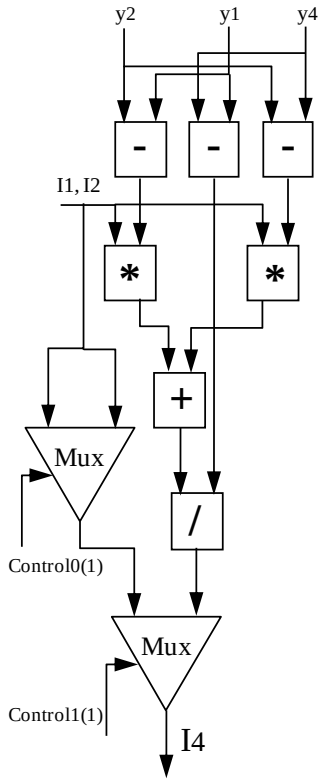
Şekil 3.19: Üçgende karşılaşılabilecek üç önemli durum

```
•
•
•
Doğru denklemlerinden elde edilen x değerlerini kontrol et
  if ((a>=minx and a<=maxx) and (b>=minx and b<=maxx)
                                and (c>=minx and c<=maxx))
    if (a==b and c>a)
      minabc = a; maxabc = c+1;
      kontrol=1;
    else if (a==b and c<a)
      minabc = c; maxabc = a+1;
      kontrol=2;
    else if (a==c and b<c)
      minabc = b; maxabc = c+1;
      kontrol=3;
    else if (a==c and b>c)
      minabc = c; maxabc = b+1;
      kontrol=4;
    else if (b==c and a>b)
      minabc = b; maxabc = a+1;
      kontrol=5;
    else if (b==c and a<b)
      minabc = a; maxabc = b+1;
      kontrol=6;
  else if ((minx>a or a>maxx))
    if (b==c)
      minabc = b; maxabc = b+1;
      kontrol=7;
    else if (b<c)
      minabc = b; maxabc = c+1;
      kontrol=8;
    else if (b>c)
      minabc = c; maxabc = b+1;
      kontrol=9;
  else if ((minx>b or  b>maxx))
    if (a==c)
      minabc = c; maxabc = c+1;
      kontrol=10;
    else if (a<c)
      minabc = a; maxabc = c+1;
      kontrol=11;
    else if (a>c)
      minabc = c; maxabc = a+1;
      kontrol=12;
  else if ((minx>c or c>maxx))
      if (a==b)
            minabc = a; maxabc = a+1;
            kontrol=13;
      else if (a<b)
            minabc = a; maxabc = b+1;
            kontrol=14;
      else if (a>b)
            minabc = b; maxabc = a+1;
            kontrol=15;
•
•
•
```

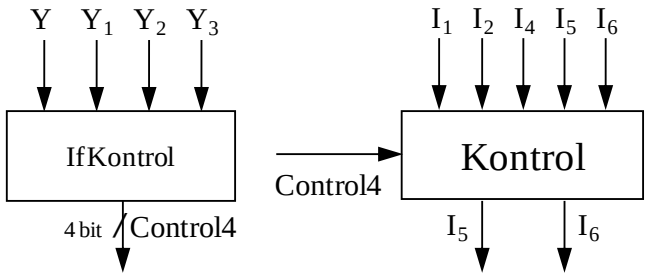
Şekil 3.20: Her doğru parçasına ait doğru denklemlerinde geçerli *y* değerine karşılık gelen *x* değerlerinin üçgenin üzerinde olup olmadığını kontrol eden algoritma parçası

Şekil 3.21'de *Renk Karar Ünitesi* blok diyagramı görülmektedir. Bu ünitede dış döngüde bir satır için başlangıç ve bitiş noktasına ait renkler indisleri ikinci bölümde verilen Denklem 2.4' e uygun olarak hesaplanmaktadır. Geçerli *y* değerinin üç adet *Cmp Ünitesi* ile köşe noktası olup olmadığı kontrol edilerek `Control1`, `Control2` ve `Control3` sinyalleri üretilir. `Control` sinyalleri ile Şekil 3.22'da görülen *Renk Genel Ünitesi* blok diyagramında yer alan *Bölme Ünite*'lerinin çalışmasını kontrol eden `RenkSecici1`, `RenkSecici2` ve `RenkSecici3` sinyallerinden hangilerinin üretileceği hangilerinin üretilmeyeceğinin kontrolünü sağlamaktadır. Burada eğer geçerli nokta bir köşe noktası ise *Bölme Ünitesi* bu değer için işlem yapmayacak ve köşe noktasına ait renk bilgisi *Renk Registerına* gecikme olmadan yazılacaktır. Ancak geçerli nokta köşe noktası değilse `Control` sinyalleri ile *Bölme Üniteleri* aktif hale gelmekte ve altı saat darbesi sonra sonucu üretmektedir. Ayrıca her hangi bir üçgen için Şekil 3.19' da görülen üçüncü olasılık ile üç renk değeri elde edilirse Şekil 3.23 a' da görülen *IfKontrol Ünitesi* ile bu durum kontrol edilmektedir. Bu ünite çıkışında `Control4` sinyali üretilir. `Control4` sinyali Şekil 3.23 b' da görülen *Kontrol Ünitesi* ile Multiplexerı kontrol ederek elde edilen üç renk bilgisinden gerekli olan renk indislerini *Renk Registerına* iletir. Ayrıca modül iç döngüye girindiğinde burada tekrar kullanılan *Renk Genel Ünitesi* her *x* değeri (tüm satır) için renk yoğunluğunu ikinci bölümde verilen Denklem 2.5' e uygun olarak hesaplar. Bu hesaplamada *Renk Karar Ünitesinde* çıkışında elde edilen iç döngü başlangıç ve bitiş değerlerine ait renk bilgileri kullanılır. Her satıra ait her bir pixel için bir renk değeri elde edildiği anda bu pixellere ait ekran hafızasındaki adres bilgilerinin hesaplanması ve bu renk değerlerinin o adreslere yazılmasını sağlamak üzere *Adres Registerı* işe koşulur.

Şekil 3.21: *Renk Karar Ünitesi* blok diyagramı



Şekil 3.22: *Renk Genel Ünitesi* blok diyagramı

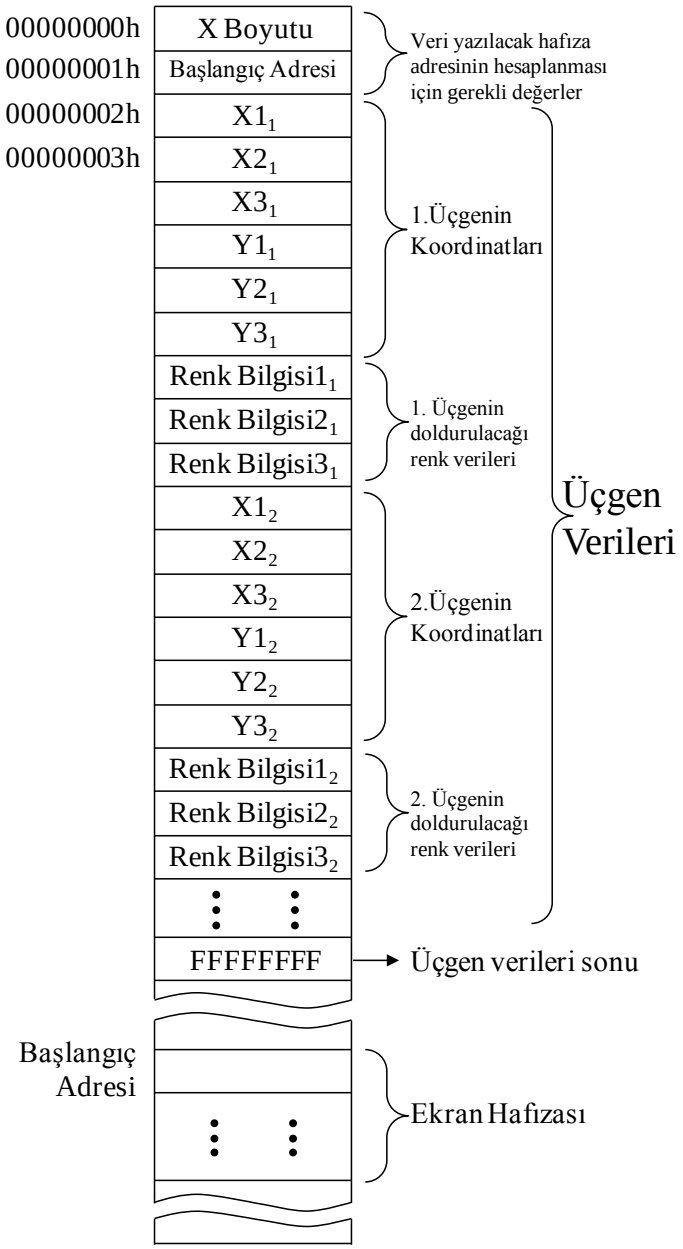


a) b)

Şekil 3.23: a) *IfKontrol Ünitesi* blok diyagramı, b) *Kontrol Ünitesi* blok diyagramı

3.1.3.1. Doğru Denklemi Modülü Hafıza Haritası ve Kontrol Ünitesi

Doğru Denklemi Modülünün kullandığı hafıza haritası Şekil 3.24'te görülmektedir. Burada *Yarı Alan Modülünün* kullandığı hafıza haritasından farklı olarak üçgenlerin her köşe noktası için farklı renk bilgileri yeralmaktadır. Tasarlanan modül üçgenin içindeki pixellerin renk değerlerini pixelin köşe noktalarına uzaklığına göre köşe noktalarının renk değerlerini kullanarak hesaplar.



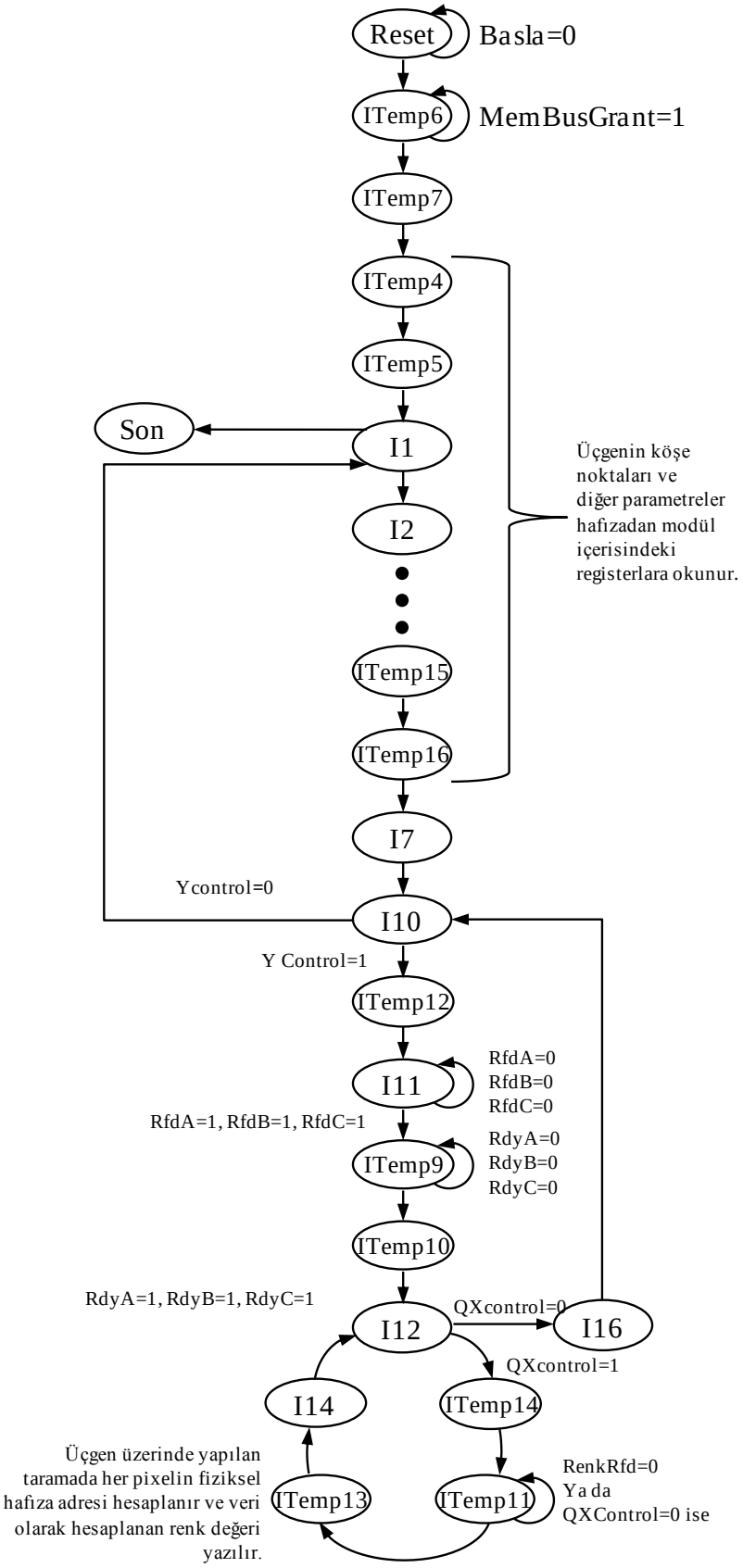
Şekil 3.24: Doğru Denklemi Modülü hafıza haritası

Şekil 3.25'te *Kontrol Ünitesinin* çalışmasını gösteren *durum* diyagramı görülmektedir. Kontrolör toplam 25 *durum*'dan oluşan bir Sonlu Durum Makinesi (Finite State Machine (FSM)) olarak tasarlanmıştır. Doğru Denklemi Modülü *Kontrol Ünitesi* başlangıçta Yarı Alan modülü ile aynı şekilde çalışır. İki modül arasındaki farklılık veri alımından itibaren başlar. Öncelikle kontrolör Yarı Alan modülü ile aynı şekilde render edilecek üçgenin grafik alanının hafıza bilgilerini (hafıza alanın başlangıç adresi ve çizim alanının genişliği) okur. Ardından kontrolör, takip eden hafıza adreslerinden üçgenin köşe noktalarının koordinatlarını ve köşe noktalarına ait üçgenin doldurulacağı

renk bilgilerini okur. Bu işlemler toplam 13 *durum* boyunca yapılır ve okunan bu veriler *Register Ünitesinde* ilgili registerlara kaydedilir. *I7 durumunda* tarama işlemi sırasında her satır için başlangıç ve bitiş *x* değerlerinin hesaplamalarında kullanılacak olan *deltax12, deltax23, deltax31, deltay12, deltay23, deltay31* fark değerleri hesaplanır ve *YSayacina* en küçük *y* değeri yüklenerek içe içe yeralan döngülerden ilki işletilmeye başlanır. İç döngü *I12*, *ITemp14*, *ITemp11*, *ITemp13* ve *I14 durumlarında* gerçekleştirilir. Dış döngü ise *I10*, *ITemp12*, *I11*, *ITemp9*, *ITemp10*, iç döngü *durum*ları (*I12*, *ITemp14*, *ITemp11*, *ITemp13*, *I14*), ve *I16* olmak üzere toplam 11 *durum*da gerçekleştirilir. *ITemp12 durumunda* tarama yapılacak her satır için üçgenin kenarlarını oluşturan üç doğru denklemini sağlayan *x* değerlerinin *Döngü Hesap Ünitesinde* hesaplanabilmesi için bu ünitede yer alan *Bölme Ünitelerini* kontrol eden `ndA`, `ndB` ve `ndC` sinyalleri aktifleştirilir. *Döngü Hesap Ünitesi* ile paralel olarak çalışmaya başlayan ve bir satır için yoğunluk hesabında kullanılacak olan renk yoğunluklarının hesaplandığı *Renk Karar Ünitesinde* yapılan kontrollerden elde edilen `Control` sinyaline göre bu ünitede yer alan *Bölüm Ünitelerinin* gerekli olanları için `RenkSec1`, `RenkSec2` ve `RenkSec3` sinyalleri de aktifleştirilmektedir. *Döngü Hesap Ünitesi*'nde yapılan işlemler ile doğru denklemlerinde geçerli *y* değeri için birer *x* değeri altı saat darbesi sonra elde edilir. *Renk Karar Ünitesinden* elde edilmesi gereken sonuç ise `Control` sinyaline bağlı olarak ya bir saat darbesi ya da altı saat darbesi sonunda üretilir. Bu işlemler için gerekli altı saat darbesi gecikme *Döngü Hesap Ünitesinde* yer alan *Bölme Ünitelerinin* kontrol sinyalleri olan `RfdA`, `RdfB`, `RdfC`, `RdyA`, `RdyB` ve `RdyC` sinyalleri kullanılarak iki *durumda* gerçeklenmiştir. *ITemp10 durumunda* iç döngü başlangıç değeri olarak *Karar Ünitesinden* elde edilen yeni minimum değeri yüklenerek iç döngünün çalışması sağlanır. *I12 durumunda XSayaci*'nın döngü kontrol sinyali `QXcontrol` 1 ise *ITemp14 durumuna* ilerlenir. *I12 durumunda* geçerli pixel bir köşe değilse yani `QXControl` 0 ise o pixele ait renk bilgisi yoğunluğu hesaplanmak üzere *Renk Genel Ünitesinde* yer alan *Bölme Ünitesinin* kontrol sinyali olan `RenkNd4` sinyali aktifleştirilir. *Bölme Ünitesi* altı saat darbesi sonra işlenmiş verinin hazır olduğunu `RenkRdy4` sinyali ile bildirir ve bu işlemler 2 *durum*da gerçekleşir. *ITemp13 durumunda* geçerli pixelin fiziksel hafiza adresi hesaplanarak hafızada bu adrese bir önceki *durumda* hesaplanmış renk bilgisinin yazılması sağlanır.

I14 durumunda ise *XSayaci* 1 arttırılarak 18. *durum*a geri döner ve bu döngü *Karar Ünitesi*'nden elde edilen maksimum değere ulaşana dek devam eder. Döngü maksimum değere ulaştığında yani `QXcontrol` 0 ise *I16 durumuna* ilerlenip *YSayaci* 1 arttırılarak döngülerin devam etmesi sağlanır. Bir üçgene ait her bir pixelin render edilmesini oluşturan iç dögüye ait işlemler on saat darbesi kadar zaman harcar.

Bir üçgenin render edilme işlemi döngüler tamalandığında bitmiş olur. Kontrolör bu aşamadan sonra yeni bir üçgenin koordinat değerlerini ve doldurulacağı renk bilgisini hafızadan okumak üzere diğer modülde olduğu gibi *I1* durumuna geri döner. Hafızadaki bütün üçgenler render edildikten sonra Yarı Alan modülü ile aynı şekilde *Stop* durumuna giderek yeni bir veri setini işlemek üzere resetlenmeyi beklemeye başlar. Kontrolör bu şekilde beklerken host bilgisayar hafızaya yeni üçgen verilerini yazarak hafızayı tazeler ve modulu tekrar başlatır.



Şekil 3.25: Modül *Kontrol Ünitesi Durum* diyagramı

4. BULGULAR

4.1. DENEY DÜZENEKLERİ VE TEST SONUÇLARI

Bu çalışmada tasarlanan modüller Rendering algoritmalarını hızlandırmak amacıyla tasarlanmıştır. Modüllerin yazılıma göre hız kazancını ölçmek için öncelikle her iki algoritma uygun bir dilde kodlanıp derlenerek algoritmaların yazılım versiyonunun verilen üçgenleri render etme süreleri ölçülmüştür. Ardından aynı üçgenlerin modüller tarafından render edilme süreleri belirlenerek hız kazancı ölçülmüştür.

Bu amaçla öncelikle Çizelge 4.1' de görülen rastgele test verileri oluşturulmuştur. Tabloda farklı büyüklüklerde altı farklı üçgenin iki boyutlu koordinat sisteminde tanımlanmış köşe noktalarının koordinat değerleri verilmiştir. Bu veriler tam sayı formatında olup her bir nokta için *x* ve *y* olmak üzere iki adet rastgele tam sayıdan oluşmaktadır. Tablonun son sütünu ise her bir üçgeni çevreleyen en küçük dikdörtgenin (bounding rectangle) pixel bazında alanını göstermektedir. Çizelge 4.2' de ise tasarlanan Doğru Denklemi modülünde boyanacak pixelin renk yoğunluğunu hesaplamak için gerekli olan üçgen köşelerine ait yine rastgele oluşturulmuş renk değerleri görülmektedir.

Çizelge 4.1: Test üçgenleri koordinatları

Üçgenler		**$P_0(X_0,Y_0)$**	**$P_1(X_1,Y_1)$**	**$P_2(X_2,Y_2)$**	***Alan**
1.	Üçgen	(1,1)	(2,5)	(5,2)	25
2.	Üçgen	(15,10)	(10,16)	(20,20)	100
3.	Üçgen	(8,8)	(9,23)	(23,9)	225
4.	Üçgen	(20,25)	(25,20)	(40,40)	400
5.	Üçgen	(10,10)	(11,50)	(50,11)	1600
6.	Üçgen	(1,1)	(2,81)	(81,2)	6400

* Üçgeni çevreleyen en küçük dörtgenin alanı (pixel olarak)

Çizelge 4.2: Üçgenlerin Köşe Koordinatlarına Ait Renk İndisleri

Üçgenler	$I_1(X_0,Y_0)$	$I_2(X_1,Y_1)$	$I_3(X_2,Y_2)$
1. Üçgen	(20)	(4)	(2)
2. Üçgen	(30)	(20)	(10)
3. Üçgen	(40)	(30)	(20)
4. Üçgen	(50)	(20)	(10)
5. Üçgen	(60)	(30)	(10)
6. Üçgen	(120)	(70)	(20)

Her iki algoritma C++ dilinde kodlanarak Visiual Studio 2008 ile derlenerek yazılım versiyonu oluşturulmuştur. Yazılım versiyonunun zamanlamasını ölçmek için Win32 tarafından sağlanan yüksek çözünürlüklü zamanlayıcı kullanılmıştır. Zamanlayıcının değeri algoritmadan önce ve sonra okunarak geçen süre mikro hassasiyetinde ölçülebilmiştir.

Oluşturulan yazılımlar Çizelge 4.3' de özellikleri verilen PC'ler üzerinde çalıştırılarak zamanlama ölçülmüştür. Çizelge 4.4'te C++' da yazılarak derlenen algoritmaların test üçgenlerini render etme süreleri verilmiştir. Çizelge 4.4' te Doğru Denklemi algoritması Yarı Alan algoritmasından daha az bir alanı taramasına rağmen Yarı Alan algoritmasından daha yavaş olduğu görülmektedir. Bunun sebebi Doğru Denklemi algoritmasında büyük ***if*** blokları yer alamasıdır. Bu bloklar işlemciyi oldukça yavaşlatmaktadırlar.

Çizelge 4.3: Çalışmada Kullanılan PC'lerin Özellikleri

Bilgisayarlar	PC1	PC2
Özellikler	Pentium4 3.00 GHz 1 GB Ram 1 MB Cache Dell	Pentium Dual-Core 2.10 GHz 3 GB Ram 2 MB Cache Asus Notebook

Çizelge 4.4: C++' da kodlanan algoritmaların test üçgenlerini render etme süreleri

Üçgenler	*Alan	Yarı Alan Algoritması		Doğru Denklemi Algoritması	
		PC1(µs)	PC2(µs)	PC1(µs)	PC2(µs)
1. Üçgen	25	16.275	15.278	27.594	20.433
2. Üçgen	100	27.505	23.485	53.858	37.823
3. Üçgen	225	32.222	31.626	64.330	44.512
4. Üçgen	400	43.109	37.566	97.960	69.675
5. Üçgen	1600	86.003	76.850	150.397	105.927
6. Üçgen	6400	187.010	188.268	335.687	231.981

* Üçgeni çevreleyen en küçük dörtgenin alanı (pixel olarak)

Modüllerin render süreleri belirlenmeden önce her iki modül hedef FPGA için sentezlenip gerçeklendikten sonra modüllerin azami çalışma frekansları (clock frekansı) ISE tasarım aracı ile bulunmuştur. Ardından modüller ISIM simülatörü yardımıyla verilen üçgenler için çalıştırılarak harcanan clock adedi belirlenmiştir. Clock adedi ve clock frekans değerleri kullanılarak Denklem 4.1'de verilen eşitlik yardımıyla modüllerin çalışma süreleri belirlenmiştir.

$$T_{MODÜL} = (1/\text{Modül Clock Frekansı}) * \text{Harcanan Clock Sayısı} \quad (4.1)$$

4.1.1. Yarı Alan Modülü

Bu bölümde Yarı Alan modülü, Xilinx firması tarafından üretilen Virtex-5 FPGA çipi için sentezlenerek çip istatistikleri ve modülün maksimum saat frekansları incelenmiştir. Çizelge 4.5' te tasarlanan Yarı Alan modülünün test üçgenlerini render etme süreleri görülmektedir. Çizelgede görüldüğü gibi üçgen boyutları büyüdükçe tasarlanan modülün render süresi de artmaktadır. Son üçgende oldukça büyük bir artış olmuştur. Bunun sebebi son iki üçgenin içinde bulunduğu bounding rectangular alanının pixel bazında artmasıdır. Diğer taraftan render edilecek grafik nesnelerinde görüntünün kalitesini arttırmak için üçgenlerin küçük boyutlarda olması tercih edilir. Bu nedenle tasarlanan modülün büyük üçgenlerde render süresinin uzun olması bir dezavantaj olmamaktadır.

Çizelge 4.5: Tasarlanan Yarı Alan modülünün test üçgenlerini render etme süreleri

Üçgenler	*Alan	Virtex-5(µs)
1. Üçgen	25	0.496
2. Üçgen	100	2.120
3. Üçgen	225	4.465
4. Üçgen	400	7.712
5. Üçgen	1600	29.721
6. Üçgen	6400	117.035

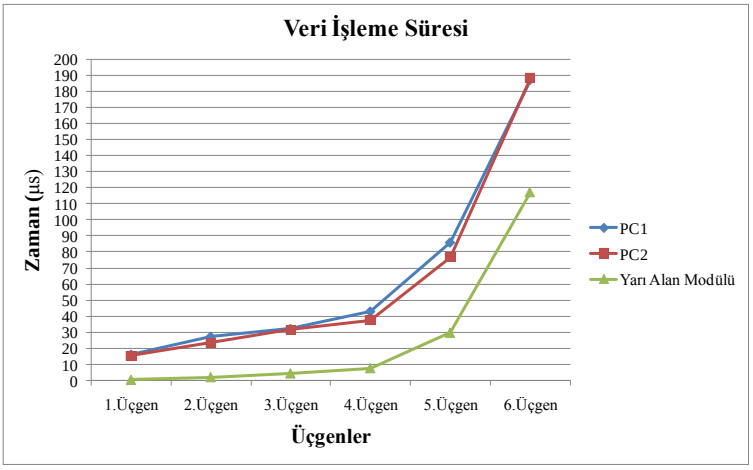
* Üçgeni çevreleyen en küçük dörtgenin alanı (pixel olarak)

Çizelge 4.6' da ise Yarı Alan modülünün Virtex-5 çipi üzerinde sentezlenmesi sonucunda oluşan çip istatistikleri verilmiştir. Çizelgede kullanılan Slice Register, Slice FFs ve IOB sayıları ile bunların kullanılma yüzdeleri görülmektedir. Bu yüzdeler incelendiğinde IOB yüzdesine göre modülün iki kopyasının bağımsız olarak tek çip içine yerleştirilebileceği ve paralel çalıştırılabileceği görülmektedir.

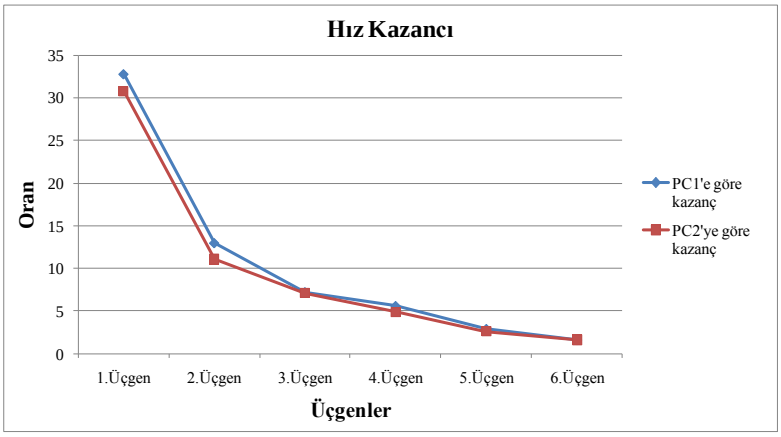
Çizelge 4.6: Yarı Alan modülü çip istatistikleri

FPGA Çip Türü	Slice Register Sayısı / %	Slice LUTs Sayısı / %	Bounded IOB Sayısı / %
Virtex-5	501 / 2	976 / 5	104 / 47

Yarı Alan algoritmasının PC1 ve PC2' deki test süreleri ile tasarlanan Yarı Alan modülünün test sürelerinin karşılaştırılması Şekil 4.1' de görülmektedir. Şekilde görüldüğü gibi tasarlanan modülün üçgenleri render ederken farklı bilgisayarlarda çalıştırılan algoritmalardan daha hızlı çalışmakta yani render işlemini daha kısa sürede tamamlamaktadır. Bu durum Şekil 4.2' de verilen kazanç grafiğinde daha net olarak görülmektedir. Grafikte görüldüğü gibi Yarı Alan modülü PC1'e göre yaklaşık 33 ile 1,5 kat arasında, PC2'ye göre ise yaklaşık 31 ile 1,5 kat arasında değişen bir oranla daha hızlı işlem yapmaktadır. Üçgen boyutları büyüdükçe modülün sağladığı avantaj gittikçe azalmaktadır. Bunun sebebi yazılımın cache bellekten yararlanıyor olmasıdır. Ancak daha önce de belirtildiği gibi render edilecek grafik nesnesini oluşturacak üçgenlerin küçük boyutlarda olması istenmektedir.



Şekil 4.1: Tasarlanan Yarı Alan modulü ile PC1 ve PC2 üzerinde çalıştırılan C++ kodunun üçgen verilerini render etme süreleri



Şekil 4.2: Tasarlanan Yarı Alan modulü ile PC1 ve PC2 üzerinde çalıştırılan C++ kodunun üçgen render etme sürelerinin oranları

4.1.2. Doğru Denklemi Modülü

Bu bölümde ise Doğru Denklemi modülü Xilinx firması tarafından üretilen Virtex-6 FPGA çipi için sentezlenerek çip istatistikleri ve modülün maksimum saat frekansları incelenmiştir. Modül testi için Çizelge 4.1' de yer alan üçgen koordinat verileri ve Çizelge 4.3' te yer alan köşe noktalarına ait rastgele oluşturulmuş renk indisleri kullanılmıştır. Çizelge 4.7' de tasarlanan Doğru Denklemi modülünün test üçgenlerini render etme süreleri görülmektedir. Ancak tasarlanan Doğru Denklemi modülü bounding rectangle tümünde değil sadece üçgenin üzerinde yani bounding rectangle alanından daha küçük bir alanda tarama yaptığı için render süreleri üçgen alanlarına bağlı olarak değişmektedir. Ayrıca modülde tamsayı bölme üniteleri kullanılması, hassas sonuç üretilemediği için, tarama yapılacak alanın daha da küçülmesine sebep olmuştur. Bu durum dördüncü üçgene ait render süresinde görülmektedir.

Çizelge 4.7: Tasarlanan Doğru Denklemi modülünün test üçgenlerini render etme süreleri

Üçgenler	*Alan	Virtex-6(µs)
1. Üçgen	25	1.631
2. Üçgen	100	5.029
3. Üçgen	225	14.174
4. Üçgen	400	11.871
5. Üçgen	1600	88.864
6. Üçgen	6400	341.782

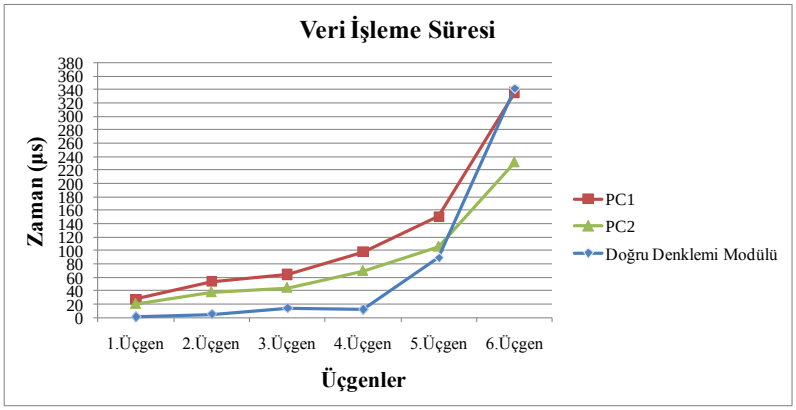
* Üçgeni çevreleyen en küçük dörtgenin alanı (pixel olarak)

Yarı Alan modülünün sentezlenmesi sonucunda oluşan istatistikler ile benzer sonuçlar Çizelge 4.8' de görülen Doğru Denklemi modülünün sentezlenmesi sonucunda da elde edilmiştir. Çizelge incelendiğinde IOB yüzdesine göre modülün iki kopyasının bağımsız olarak tek çip içine yerleştirilebileceği ve paralel çalıştırılabileceği görülmektedir.

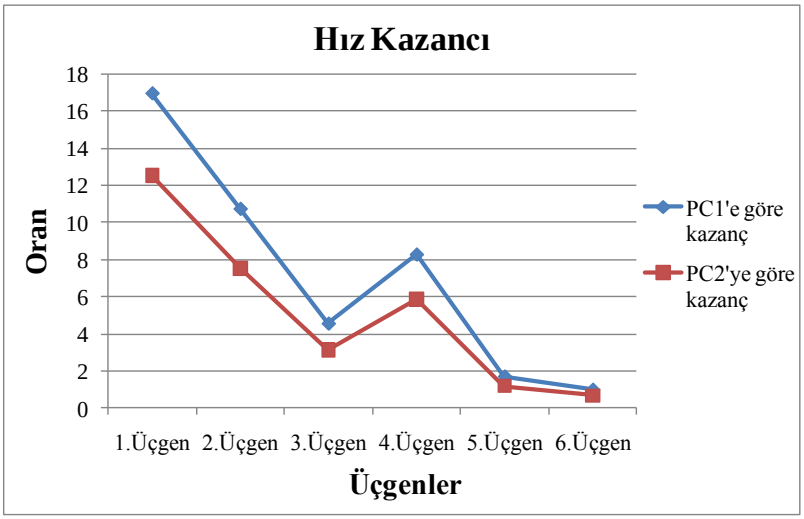
Çizelge 4.8: Doğru Denklemi modülü çip istatistikleri

FPGA Çip Türü	Slice Register Sayısı / %	Slice LUTs Sayısı / %	Slice Occupied Sayısı / %	Bounded IOB Sayısı / %
Virtex-6	2412 / 02	7801 / 16	2577 / 22	140 / 43

Doğru Denklemi algoritmasının PC1 ve PC2' deki test süreleri ile tasarlanan Doğru Denklemi modülünün test sürelerinin karşılaştırılması Şekil 4.3' te görülmektedir. Şekilde görüldüğü gibi tasarlanan modülün üçgenleri render ederken farklı bilgisayarlarda çalıştırılan algoritmalardan daha hızlı çalışmakta yani render işlemini daha kısa sürede tamamlamaktadır. Bu durum Şekil 4.4' te verilen kazanç grafiğinde görülmektedir. Grafikte görüldüğü gibi tasarlanan Doğru Denklemi modülü PC1'e göre yaklaşık 17 ile 1 kat arasında, PC2'ye göre ise yaklaşık 12,5 ile 0,5 kat arasında değişen bir oranla daha hızlı işlem yapmaktadır. Yarı Alan modülünde karşılaşıldığı gibi Doğru Denklemi modülünde de üçgen boyutları büyüdükçe modülün sağladığı avantaj gittikçe azaldığı sorunuyla karşılaşılmıştır. Ancak grafiklerinde nesneler kaliteli görüntü oluşturmak için küçük boyutlu üçgenlerle matematiksel olarak modellendiği için bu durum tasarımların kullanılabilirliğini etkilememektedir.



Şekil 4.3: Tasarlanan Doğru Denkelemi modulü ile PC1 ve PC2 üzerinde çalıştırılan C++ kodunun üçgen verilerini render etme süreleri



Şekil 4.4: Tasarlan Doğru Denklemi modulü ile PC1 ve PC2 üzerinde çalıştırılan C++ kodunun üçgen render etme sürelerinin oranları

5. TARTIŞMA VE SONUÇ

Yazılım ve donanım olarak farklı şekillerde gerçeklenen Rendering işleminde yazılımsal çözümler genellikle zaman problemi olmayan ve çok hassas hesaplamaya gerek duyulmayan durumlarda kullanılmaktadır. Donanımsal çözümlerde daha hızlı ve birçok hesaplama yapılması gereken yerlerde tercih edilmektedir. Ancak, grafik işlemlerinde, grafiklerdeki nesne sayısı ve bu nesneleri tanımlamada kullanılan üçgen sayısı arttıkça, grafiklerin render edilmesi çok daha uzun sürebilmektedir. Bu durumda genel amaçlı bilgisayarlar yetersiz kalmakta ve grafik nesnelerinin hesaplanması çok uzun süreler almaktadır. Bu duruma çözüm olarak farklı yaklaşımlar geliştirilmiştir. Bu yaklaşımlardan bazıları; gelişmiş grafik kartların, grafik işlemler için tasarlanmış özel amaçlı bilgisayarların, daha hızlı işlemcilerin ya da paralel işlemcilerin kullanılmasıdır (Koyuncu, 2008).

Bu çalışmada yukarıda sayılan çalışmalara bir alternatif olarak FPGA yongaları üzerinde çalışabilecek tam sayı tabanlı iki donanım modülü tasarlanmıştır. Böylelikle grafik rendering işleminin, daha ucuz, daha hızlı ve daha esnek bir şekilde hesaplanması hedeflenmiştir.

Çalışmada ilki yarı alan denkleminden ikincisi ise doğru denkleminden yararlanılarak oluşturalan tamsayı tabanlı iki modül tasarlanmıştır. Tasarlanan modüller, rastgele oluşturulan farklı büyüklüklerde test üçgenleri kullanılarak test edilmiştir. Modüllerin tasarımında temel alınan algoritmalar C++ dilinde derlenerek, ayrıca yazılım versiyonları da oluşturulmuştur. Yazılım versiyonları farklı genel amaçlı bilgisiayarlarda çalıştırılarak aynı test üçgenlerini test etme süreleri ölçülmüştür. Yazılım versiyonları ile FPGA üzerinde çalışan modüllerin test üçgenlerini render etme süreleri karşılaştırıldığında Yarı Alan modülünün 31 ile 1.5 arasında, Doğru Denklemi modülünün ise 17 ile 0.5 arasında daha hızlı render işlemi gerçekleştirildiği görülmüştür.

Modüllerin hız kazançları daha detaylı incelendiğinde küçük boyutlu üçgenlerde modüllerin hız kazancının daha yüksek olduğu tespit edilmiştir. Rendering işleminde grafik objelerin daha küçük üçgenlerle tanımlanması daha gerçekçi görüntülerin oluşturulmasına yardımcı olduğundan bu durum modüllerin önemini daha da arttırmaktadır.

Kazançlar tek bir FPGA üzerinde tek bir modül çalıştırılarak tespit edilmiştir. Bir FPGA çipine birden fazla modülün uygun şekilde yerleştirilmesiyle ya da birden fazla FPGA çipinde modülün birden fazla kopyasını çalıştırılmasıyla hız kazancının katlanarak artacağı düşünülmektedir.

5.1. GELECEKTE YAPILABİLECEK ÇALIŞMALAR

Modüller tek bir FPGA çipi üzerinde tek modül olarak çalışacak biçimde tasarlanmıştır. Çip istatistiklerine bakıldığında bir çip içine birden fazla modül kopyasının rahatça sığacağı görülmektedir. Hız kazancını arttırmak için bir çipe birden fazla modül yerleştirirlerek denenebilir. Ayrıca birden fazla FPGA çipinde birden fazla modül paralel olarak çalışmasını sağlamak için grafik programlarından gelen dönüşüm isteklerinin uygun şekilde sıralayıp çiplere dengeli bir şekilde dağıtan bir arayüz programı geliştirilebilir. Bunlara ek olarak Doğru Denklemi modülde yer alan kullanıma hazır tamsayı *Bölme Üniteleri* yerine daha kısa sürede sonuç üretecek bölme üniteleri tasarlanarak modülün harcadığı süre azaltılabilir ve belirlenen tarama alanı hassasiyeti arttırılabilir.

KAYNAKLAR

Anonim, 2011, http://en.wikipedia.org/wiki/Field-programmable_gate_array [Ziyaret tarihi 11 Ağustos 2011].

Anonim, 2011, http://www.bilesim.com/ [Ziyaret Tarihi 09 Ağustos 2011].

Anonim, 2011, http://www.xilinx.com/support/documentation/data_sheets [Ziyaret Tarihi 10 Ağustos 2011].

Anonim, 2009, http://www.sgi.com/ [Ziyaret tarihi: 17 Mart 2009].

Anonim, 2010, http://mech.fsv.cvut.cz/~dr/papers/Habil/img989.gif [Ziyaret Tarihi: 10 Eylül 2010].

Anonim, 2010, http://www.pixar.com/howwedoit/index.html [Ziyaret tarihi: 20 Şubat 2010].

Anonim, 2009, http://www.vision.caltech.edu/ [Ziyaret Tarihi 12 Ekim 2009].

Beeckler, J., S., Gross, W., J., 2005, FPGA Particle Graphics Hardware, Proceedings of the *13th Annual IEEE Symposium on Field-Programmable Custom Computing Machines.*

Beuchat, J., L., 2003, Modular Multiplication for FPGA Implementation of the IDEA Block Cipher, *Proceedings of ASAP Application Specific Systems Architectures and Processors IEEE*, Silverspring, MD.

Bishop, B., Kelliher, T., P., 2003, Specialized Hardware for Deformable Object Modeling, *IEEE Transactions on circuits and Systems for Video Technology*, Vol. 13, No. 11.

Borgatti, M., Lertora, F., Foret, B., Cali, L., 2002, A Reconfigurable System Featuring Dynamically Extensible Embedded Microproccessor, FPGA and Customizable I/O, *IEEE Custom Integrated Circuits Conference, 13-16.*

Capens, N., 2009, http://www.devmaster.net/forums/showthread.php?t=1884, [Ziyaret Tarihi 09-11-2009].

Çavuşlu, M., Dikmeşe, A., Şahin, S., Küçük, S. K. ve Kavak, A., 2006, Akıllı Anten Algoritmalarının IEEE 754 Kayan Sayı Formatı ile FPGA Tabanlı Gerçeklenmesi ve Performans Analizi, *III.URSI-Türkiye'2006*, pp.610-612.

Chodowiec, P., Gaj, K., 2003, Very compact FPGA Implementation of the AES, *Proceedings of CHES 2003, Lecture Notes in Computer Science*, vol. 2779, Springer, Berlin, pp. 319–333.

Dillon, T., 2001, Two Virtex-II FPGAs Deliver Fastest, Cheapest, Best High-Performance Image Processing System, *Xilinx Xcell Journal*, 41.

Eadie, D., Shevlin, F., Nisbet, A., 2002, Correction of Geometric Image Distortion Using FPGAs, Optical Mefrologv, *Imaging, and Machine Vision Conference*, Galway, IRL.

Fender, J., Rose, J., 2003, AHigh speed Ray Tracing Engine Built on a Field-Programmable System, *IEEE*.

Gloster, C., Sahin, I., 2001, Floating-Point Modules Targeted for Use with RC Compilation Tools, *Earth Science Technology Conference (ESTC)*, College Park, MD.

Hearn, D., Baker M.P., 2003, Computer Graphics with OpenGL 3E, *Prentice Hall.*

Hu, H., Jin, T., Zhang, X., Lu, Z. and Qian, Z. 2006, A Floating-point Coprocessor Configured by a FPGA in a Digital Platform Based on Fixed-point DSP for Power Electronics, *IEEE IPEMC*.

Koca,H., 2007, *Robot Manipülatör Denetimi,*Yüksek Lisans, Gazi Üniversitesi.

Koyuncu, İ., 2008, *Grafik Sistemleri için FPGA Cihazlarında Çalışmak Üzere Tasarlanmış Matris Çarpım Motoru*, Yüksek Lisans Tezi, Düzce Üniversitesi.

Koyuncu, İ., Şahin, İ. 2007, Generic Fpga Modules for Integer 2D and 3D Transformations, 12th. Conference for Computer Aided Engineering and System Modeling with Exhibition, Antalya.

Majer M., Wildermann, S., Angermeier, J., Hanke, S., Teich, J., 2004, Co-Design Architecture and Implementation for Point-Based Rendering on FPGAs, *The 19th IEEE/IFIP International Symposium on Rapid System Prototyping*

McKenna, M. ve Wilamowski, M., 2001, Implementing a Fuzzy System on a Field Programmable Gate Array. *International Joint Colnference on Neural Networks*, p.p. 189-194.

Rincon, F., Teres, L., 1998, Reconfigurable Hardware Systems, *International Semiconductor Conference*, vol. 1, pp. 45-54.

Rouvroy, G., Standaert, F., Quisquater, J. and Legat, J. 2004, Compact and Efficient Encryption/decryption Module for FPGA Implementation of the AES Rijndael Very Well Suited for Small Embedded Applications, *in: Proceedings of ITCC 2004*, Las Vegas, April 5–7,

Şahin, I., Gloster, C. ve Doss, C., 2000, Feasibility of Floating-Point Arithmetic in Reconfigurable Computing Systems, *Military and Aerospace Applications of Programmable Devices and Technology Conference*, Washington, DC.

Sırmaçek, B., 2007, *FPGA İle Mobil Robot İcin Öğrenme Algoritması Modellenmesi*, Yüksek Lisans Tezi, Yıldız Teknik Üniversitesi.

Sridharan, K., Priya, T.K., 2007, A Hardware Accelerator and FPGA Realization for Reduced Visibility Graph Construction Using Efficient Bit Representations, *IEEE Transactions on Industrial Electronics*, Vol. 54, No. 3.

Şahin, I., Gloster, C.S., 2005b, FPGA Tabanlı CCM'ler İçin Genel Amaçlı Bir Arayüz, *Bilimde Modern Yöntemler Sempozyumu*, Kocaeli.

Şahin, İ., Gloster, C.S., 2005a, Evaluation of Ic Physical Design Optimization Algorithms For Acceleration Using Fpga-Based Custom Computing Machines, *İleri Teknolojiler Sempozyumu*, Konya.

Şahin, İ., Koyuncu, İ. , 2008, Grafik Sistemleri için FPGA Tümdevrelerinde Çalışmak Üzere Tasarlanmış Matris Çarpım Motoru, *SAÜ. Fen Bilimleri Dergisi*.

Uzun,I., S., Amira, A., Bouridane, A., 2005, FPGA implementations of fast Fourier transforms forreal-time signal and image processing, IEE Proceedings ofImage and Signal Processing, p.p.: 283-296

Visser, S.J., Dawood, A.S. ve Williams, J.A. 2002, *FPGA Based Real-time Adaptive Filtering for Space Applications*, in: Proceedings of IEEE International Conference on Field-Programmable Technology, pp. 322–326.

Walke, R.L., Smith, R.W.M., Lightbody, G. 2000, *20 GFLOPS QR Processor on a Xilinx Virtex-E FPGA*, SPIE, San Diego, U.S.A.

Yılmaz, N., 2008, *Alan Programlamalı Kapı Dizileri (FPGA) Üzerinde Bir YSA'nın Tasarlanması ve Donanım Olarak Gerçekleştirilmesi*, Yüksek Lisans Tezi, Selçuk Üniversitesi.

EKLER

EK-A: YARI ALAN MODÜLÜ TASARLANIRKEN KULLANILMAK ÜZERE HAZIRLANAN PROGRAM KODU

```
#include<iostream>
#include <fstream>
#include <iomanip>
#include"Rendering.h"
using namespace std;

int MyMin(int x, int y, int z)
{return min(min(x,y),z);}
int MyMax(int x, int y, int z)
{return max(max(x,y),z);}
void Render(Ucgen U)
{
  int y1 = U.V1.y;
  int y2 = U.V2.y;
  int y3 = U.V3.y;
  int x1 = U.V1.x;
  int x2 = U.V2.x;
  int x3 = U.V3.x;

  int Dx12 = x1 - x2;
  int Dx23 = x2 - x3;
  int Dx31 = x3 - x1;
  int Dy12 = y1 - y2;
  int Dy23 = y2 - y3;
  int Dy31 = y3 - y1;

  int minx = (int)MyMin(x1, x2, x3);
  int maxx = (int)MyMax(x1, x2, x3);
  int miny = (int)MyMin(y1, y2, y3);
  int maxy = (int)MyMax(y1, y2, y3);

  int C1 = Dy12 * x1 - Dx12 * y1;
  int C2 = Dy23 * x2 - Dx23 * y2;
  int C3 = Dy31 * x3 - Dx31 * y3;
```

```
  int Cy1 = C1 + Dx12 * miny - Dy12 * minx;
  int Cy2 = C2 + Dx23 * miny - Dy23 * minx;
  int Cy3 = C3 + Dx31 * miny - Dy31 * minx;

  for (k=1; k<maxy; k++)
  {
    for (l=1; l<maxx; l++)
      Dizi[k][l]=0 ;
  }
 for(int y = miny; y < maxy; y++)
 {
   int Cx1 = Cy1;
   int Cx2 = Cy2;
   int Cx3 = Cy3;
   for(int x = minx; x < maxx; x++)
   {
     if(Cx1 > 0 && Cx2 > 0 && Cx3 > 0)
      {
       Dizi[i][j]=1;
       Cikis <<"x= "<< x << " y = "<< y << "true "<<Dizi[i][j]<<endl;
      }
     Cx1 -= Dy12;
     Cx2 -= Dy23;
     Cx3 -= Dy31;
   }
    Cy1 += Dx12;
    Cy2 += Dx23;
    Cy3 += Dx31;
 }
}
void main(int argc, char *argv[])
{
      Ucgen F;
      F.V1.x=1;
      F.V1.y=1;
      F.V2.x=2;
      F.V2.y=48;
      F.V3.x=48;
      F.V3.y=2;
      Render(F);
```

```
        cin.get();
        cin.get();

}
```

EK-B: DOĞRU DENKLEMİ MODÜLÜ TASARLANIRKEN KULLANILMAK ÜZERE HAZIRLANAN PROGRAM KODU

```
#include<iostream>
#include <fstream>
#include <iomanip>
#include <bitset>
#include"renderrr.h"
using namespace std;

int MyMin(int x, int y, int z)
{return min(min(x,y),z);}
int MyMax(int x, int y, int z)
{return max(max(x,y),z);}
void Render(Ucgen U)
{
 ofstream Cikis;
 Cikis.open("deneme.txt");
 int Dizi1[50][50];
 int Dizi2[500][500];
 int y1 = U.V1.y;
 int y2 = U.V2.y;
 int y3 = U.V3.y;
 int x1 = U.V1.x;
 int x2 = U.V2.x;
 int x3 = U.V3.x;
 int I1 = U.V1.I;
 int I2 = U.V2.I;
 int I3 = U.V3.I;
 // Deltalar
 int Dx12 = x1 - x2;
 int Dx23 = x2 - x3;
 int Dx31 = x3 - x1;
 int Dy12 = y1 - y2;
 int Dy23 = y2 - y3;
 int Dy31 = y3 - y1;
 // Üçgeni Çevreleyen en küçük dörtgen
 int minx = (int)MyMin(x1, x2, x3);
 int maxx = (int)MyMax(x1, x2, x3);
```

```
int miny = (int)MyMin(y1, y2, y3);
int maxy = (int)MyMax(y1, y2, y3);

int k,l;
for (k=0; k<maxy+1; k++)
{
     for (l=0; l<maxx+1; l++)
     {
           Dizi1[k][l]=0 ;
           Dizi2[k][l]=0 ;
     }
}
int i=miny;
int I4,I5,IP;
int kontrol=0;
for(int y = miny; y <= maxy; y++)
{
 // Yatay tarama için başlangıç değerleri
 int k,l,m,a,b,c;
 r=(((float)(Dx12*(y1-y))-(float)(Dy12 * x1))/(float)(-1* Dy12));
 s=(((float)(Dx23*(y2-y))-(float)(Dy23 * x2))/(float)(-1* Dy23));
 t=(((float)(Dx31*(y3-y))-(float)(Dy31 * x3))/(float)(-1* Dy31));

 a=(((Dx12*(y1-y))-(Dy12 * x1))/(-1* Dy12));
 b=(((Dx23*(y2-y))-(Dy23 * x2))/(-1* Dy23));
 c=(((Dx31*(y3-y))-(Dy31 * x3))/(-1* Dy31));

 if((r-a)>= 0.5)
 {a++;}
 if((s-b)>=0.5)
 {b++;}
 if((t-c)>= 0.5)
 {c++;}
 int minabc;
 int maxabc;
 if((minx<=a && a<=maxx)&&(minx<=b && b<=maxx)&&(minx<=c && c<=maxx))
 {
  if (a==b && c>a)
  {
   minabc = a;
```

```
 maxabc = c+1;
 kontrol=1;
}
else if (a==b && c<a)
{
 minabc = c;
 maxabc = a+1;
 kontrol=2;
}
else if (a==c && b<c)
{
 minabc = b;
 maxabc = c+1;
 kontrol=3;
}
else if (a==c && b>c)
{
 minabc = c;
 maxabc = b+1;
 kontrol=4;
}
else if (b==c && a>b)
{
 minabc = b;
 maxabc = a+1;
 kontrol=5;
}
else if (b==c && a<b)
{
 minabc = a;
 maxabc = b+1;
 kontrol=6;
}
}
   else if ((minx>a || a>maxx))
   {     if (b==c)
         {     minabc = b;
               maxabc = b+1;
               kontrol=7;
         }
```

```
      else if (b<c)
      {     minabc = b;
            maxabc = c+1;
            kontrol=8;
      }
      else if (b>c)
      {     minabc = c;
            maxabc = b+1;
            kontrol=9;        }
}
else if ((minx>b || b>maxx))
{
      if (a==c)
      {     minabc = c;
            maxabc = c+1;
            kontrol=10;
      }
      else if (a<c)
      {     minabc = a;
            maxabc = c+1;
            kontrol=11;
      }
      else if (a>c)
      {     minabc = c;
            maxabc = a+1;
            kontrol=12;
      }

}
else if ((minx>c || c>maxx))
{
      if (a==b)
      {     minabc = a;
            maxabc = a+1;
            kontrol=13;
      }
      else if (a>b)
      {     minabc = b;
            maxabc = a+1;
            kontrol=14;
```

```
        }
        else if (a<b)
        {     minabc = a;
              maxabc = b+1;
              kontrol=15;
        }
}
switch (kontrol){
        case 1:
              I4=I2;
              I5=(((y-y3)*I1)/(y1-y3)) + ((y1-y)*I3)/(y1-y3);
              break;
        case 2:
              I4=(((y-y3)*I1)/(y1-y3))+((y1-y)*I3)/(y1-y3);
              I5=I2;
              break;
        case 3:
              I4=(((y-y3)*I2)/(y2-y3))+((y2-y)*I3)/(y2-y3);
              I5=I1;
              break;
        case 4:
              I4=I1;
              I5=(((y-y3)*I2)/(y2-y3))+((y2-y)*I3)/(y2-y3);
              break;
        case 5:
              I4=I3;
              I5=(((y-y2)*I1)/(y1-y2))+((y1-y)*I2)/(y1-y2);
              break;
        case 6:
              I4=(((y-y2)*I1)/(y1-y2))+((y1-y)*I2)/(y1-y2);
              I5=I3;
              break;
        case 7:
              I4=I3;
              I5=I3;
              break;
        case 8:
              I4=(((y-y3)*I2)/(y2-y3))+((y2-y)*I3)/(y2-y3);
              I5=(((y-y3)*I1)/(y1-y3))+((y1-y)*I3)/(y1-y3);
              break;
```

```
            case 9:
                  I4=(((y-y3)*I1)/(y1-y3))+((y1-y)*I3)/(y1-y3);
                  I5=(((y-y3)*I2)/(y2-y3))+((y2-y)*I3)/(y2-y3);
                  break;
            case 10:
                  I4=I1;
                  I5=I1;
                  break;
            case 11:
                  I4=(((y-y2)*I1)/(y1-y2))+((y1-y)*I2)/(y1-y2);
                  I5=(((y-y3)*I1)/(y1-y3))+((y1-y)*I3)/(y1-y3);
                  break;
            case 12:
                  I4=(((y-y3)*I1)/(y1-y3))+((y1-y)*I3)/(y1-y3);
                  I5=(((y-y2)*I1)/(y1-y2))+((y1-y)*I2)/(y1-y2);
                  break;
            case 13:
                  I4=I2;
                  I5=I2;
                  break;
            case 14:
                  I4=(((y-y3)*I2)/(y2-y3))+((y2-y)*I3)/(y2-y3);
                  I5=(((y-y2)*I1)/(y1-y2))+((y1-y)*I2)/(y1-y2);
                  break;
            case 15:
                  I4=(((y-y2)*I1)/(y1-y2))+((y1-y)*I2)/(y1-y2);
                  I5=(((y-y3)*I2)/(y2-y3))+((y2-y)*I3)/(y2-y3);
                  break;
            default:
                  I4=0;
                  I5=0;
      }
      int j=minabc;
        for(int x = minabc; x < maxabc; x++)
        {
                        IP=(( ((maxabc-x)*I4) / (maxabc-minabc) )+(
((x-minabc)*I5) / (maxabc-minabc) ));
                        Dizi2[i][j]=IP;
                        j++;
        }
```

```
            i++;
      }
        for (k=0; k<maxy+1; k++)
        {
            for (l=0; l<maxx+1; l++)
                        Cikis << setw (5)<< Dizi2[k][l] ;
            Cikis<<"\n";
        }
}
void main(int argc, char *argv[])
{

        Ucgen F;
        F.V1.x=1;
        F.V1.y=1;
        F.V2.x=2;
        F.V2.y=5;
        F.V3.x=5;
        F.V3.y=2;

        F.V1.I=20;
        F.V2.I=4;
        F.V3.I=2;
        Render(F);

        cin.get();
        cin.get();


}
```

EK-C: BİLGİSAYAR İŞLEM SÜRESİ HESAPLANMASI İÇİN HAZIRLANAN PROGRAM KODU

Aşağıda algoritması ve C++ kodu verilen program deneylerde kullanılan PC'lerde yazılım programın veri işleme sürelerini belirlemek amacıyla oluşturulmuştur.

Adım 1 Başla.

Adım 2 CPU hızını ve veri dosyası adını gir.

Adım 3 Veriyi oku.

Adım 4 CPU saat değerini oku ve kaydet.

Adım 5 Verileri istenen şekilde işle.

Adım 6 CPU saat değerini yeniden oku ve kaydet.

Adım 7 CPUsaat verilerini kullanarak işlem süresini hesapla.

Adım 8 Sonucu ekrana yaz.

```
#include <fstream>
#include <iostream>
#include <iomanip>
#include"Rendering.h"

using namespace std;

#define CpuId __asm __emit 0fh __asm __emit 0a2h
#define RdtSc __asm __emit 0fh __asm __emit 031h


unsigned *GetCycles()
{
      unsigned *cycles;
      cycles = new unsigned[2];

      unsigned cycles_high=0, cycles_low=0;
      __asm
      {
            pushad
            CpuId
```

```
            RdtSc
            mov     cycles_high, edx
            mov     cycles_low, eax
            popad
      }

      cycles[0] = cycles_high;
      cycles[1] = cycles_low;
      return cycles;
}


double GetMsec (unsigned *Cycle1, unsigned *Cycle2, int CPU)
{
      unsigned __int64 temp_cycles1=0, temp_cycles2=0;
      __int64 total_cycles=0;

      temp_cycles1 = ((unsigned __int64)Cycle1[0] << 32) | Cycle1[1];
      temp_cycles2 = ((unsigned __int64)Cycle2[0] << 32) | Cycle2[1];

      total_cycles = temp_cycles2 - temp_cycles1;

      return double(total_cycles)/double(CPU);
}

void main(int argc, char *argv[])
{
      unsigned    *T0,*T1;
      T0 = new unsigned[2];
      T1 = new unsigned[2];

      int CPU;

      if (argc != 2)
      {
            printf("Usage: <CPU Speed (Mhz)>\n");
            exit(1);
      }

      CPU = atoi(argv[1]);
```

```
	cout << "Baslamak icin hazir. Devam etmek icin enter tusuna
basiniz\n";
	cin.get();
//ZAMANLAMASI ÖLÇÜLECEK PROGRAMIN YAZILACAĞI YER//
	//Read the initial time info
	T0 = GetCycles();

	Render(A);

	//Read the final time info
	T1 = GetCycles();

	//Print the result
	cout << setiosflags(ios::showpoint|ios::fixed);
	cout << setprecision(10);
	cout << "\nExecution Time = " << setw(12) <<
GetMsec(T0,T1,CPU)/1000000 << " sec.\n";
	cout << "Execution Time = " << setw(12) <<
GetMsec(T0,T1,CPU)/1000 << " msec.\n";
	cout << "Execution Time = " << setw(12) << GetMsec(T0,T1,CPU) <<
" usec.\n";

	cin.get();
	cin.get();
}
```

ÖZGEÇMİŞ

Kişisel Bilgiler	
Soyadı, adı	: OK, Emel
Uyruğu	: T.C
Doğum tarihi ve yeri	: 17/03/1984, İstanbul
Medeni Durum	: Bekar
Telefon	: 0380 411 65 39
E-mail	: abzahemel@hotmail.com

Eğitim Derecesi	**Eğitim Birimi**	**Mezuniyet**
Yüksek Lisans	: Düzce Üniversitesi / Elektrik Eğitimi Anabilim Dalı	2011
Lisans	: Marmara Üniversitesi Teknik Eğitim Fakültesi / Elektronik ve Bilgisayar Bölümü / Bilgisayar ve Kontrol Öğretmenliği	2008
Lise	: Tuzla Anadolu Teknik	2003

YABANCI DİL

İngilizce: Upper Intermediate (Yüksek Orta Seviye)

BİLGİSAYAR

VHDL, C++, Visual Basic, Java, OpenGL, Pascal, Microsoft Office

İLGİ ALANLARI

Bilgisayar, gezi, kitap, internet, spor, müzik.