Üç Cisim Problemi ve Roket Simülasyonu Proje Raporu

Eray Naldöken 191201059

İçindekiler

1 Hedeflenen Programın Kısa Özeti

Bu proje, üç cisim problemini ele alan ve roketlerin hareketlerini simüle eden bir sistemin C++ ile gerçeklenmesini hedeflemektedir. Program, nesne yönelimli programlama (OOP) prensiplerine uygun olarak geliştirilmiş, cisimlerin ve roketlerin yörünge hareketlerini hesaplamakta ve bunları bir grafiksel arayüz üzerinde (canvas) görselleştirmektedir.

2 Nesne Yönelimli Programlama Anlayışında Parçalama

Program, OOP prensiplerine uygun şekilde modüler olarak tasarlanmıştır. Her bir modül, belirli bir sorumluluğa sahiptir ve programın farklı parçalarını temsil eder. Program şu bileşenlerden oluşmaktadır:

- `vector` **Modülü**: 2 boyutlu vektör işlemlerini gerçekleştiren temel sınıf. Vektörlerin toplama, çıkarma, skaler çarpma gibi matematiksel işlemlerini ve büyüklük hesaplamalarını içermektedir.
- `body` **Modülü**: Noktasal cisimlerin hareketlerini tanımlayan sınıftır. Cisimlerin kütle, konum ve hız bilgilerini içermektedir ve Newton'un hareket yasalarına göre konum ve hız güncellemesi yapmaktadır.
- `rocket` **Modülü**: Roket hareketini modelleyen sınıftır. `body` sınıfından türetilmiştir ve ek olarak roketin itki kuvveti ve kütle kaybını hesaplamaktadır.
- `universe` **Modülü**: Sistemdeki tüm cisim ve roketlerin etkileşimlerini yöneten sınıftır. Cisimler arasındaki kütle çekim kuvvetlerini hesaplamakta ve sistemin zaman adımına göre hareketini güncellemektedir.
- `canvas` **Modülü**: Simülasyonun grafiksel çıktısını sağlayan sınıftır. Cisim ve roketlerin yörüngelerini farklı renklerde çizerek kullanıcıya görselleştirme imkânı sunmaktadır.

Program, tek bir kişi tarafından gerçekleştirilmiştir.

3 Sınıf, Nesne, Değişken ve Üye Fonksiyonların Tanımları

Bu bölümde programda kullanılan sınıflar, üye değişkenler ve fonksiyonlar detaylı olarak tanıtılmış ve her birinin kullanım amacı açıklanmıştır.

3.1 `vector` Sınıfı

Amaç: İki boyutlu vektör işlemlerini gerçekleştirmek için kullanılır. Cisimlerin konum ve hız gibi vektörel büyüklüklerinin hesaplanmasında önemli rol oynar.

Üye Değişkenler:

- `double x, y`: Vektörün x ve y bileşenlerini temsil eder.

Fonksiyonlar:

- `vector operator+(const vector&)`: İki vektörü toplar. Cisimlerin hız ve konum değişimlerini hesaplamak için kullanılır.
- `vector operator-(const vector&)`: İki vektörü çıkarır. Cisimler arasındaki mesafeyi ve yönü belirlemek için kullanılır.
- `vector operator*(double)`: Vektörü bir skaler ile çarpar. İvmeyi, hızı veya konumu belirli bir zaman adımına göre güncellemek için kullanılır.
- `double find_magnitude()`: Vektörün büyüklüğünü hesaplar. İki cisim arasındaki mesafeyi veya bir cismin hız büyüklüğünü bulmak için kullanılır.

3.2 `body` Sınıfı

Amaç: Noktasal cisimlerin kütle, konum ve hız bilgilerini tutar ve hareketlerini tanımlar. Newton'un hareket yasalarına göre cisimlerin hareketini günceller.

Üye Değişkenler:

- `double mass`: Cismin kütlesini temsil eder.
- `vector position`: Cismin anlık konumunu temsil eder.
- `vector velocity`: Cismin anlık hızını temsil eder.

Fonksiyonlar:

- `void update(const vector&, double)`: Cismin hızını ve konumunu, kendisine uygulanan net kuvvete ve zaman adımına göre günceller. Hareket denklemlerini kullanarak cismin yörüngesini hesaplar.
- `vector getPosition()`: Cismin mevcut konumunu döner. Görselleştirme ve kuvvet hesaplamalarında kullanılır.

3.3 `rocket` Sınıfı (Kalıtım ile `body` Sınıfından Türetilir)

Amaç: Roketlerin hareketini modellemek için kullanılır. Roketlerin itki kuvveti ve kütle kaybı gibi ek özelliklerini tanımlar.

Ek Üye Değişkenler:

- `double exhaustVelocity`: Roketten püskürtülen gazın hıza katkısını temsil eder.
- `double burnRate`: Roketin birim zamanda yaktığı yakıt miktarını temsil eder.

Fonksiyonlar:

- `void update(const vector&, double)`: Roketin hızını, konumunu ve kütlesini günceller. İtki kuvvetini hesaba katar ve yakıt tükendiğinde kütleyi sıfıra indirir.

3.4 `universe` Sınıfı

Amaç: Sistemdeki tüm cisim ve roketlerin etkileşimlerini yönetir. Cisimler arasındaki kütle çekim kuvvetlerini hesaplar ve tüm sistemin hareketini simüle eder.

Üye Değişkenler:

- `std::vector<body*> bodies`: Sistemde bulunan tüm cisim ve roketlerin referanslarını tutar.

Fonksiyonlar:

- `void insertBody(body&)`: Sisteme yeni bir cisim veya roket ekler. Simülasyonda yer alacak nesneleri tanımlamak için kullanılır.
- `void run(double)`: Sistemdeki tüm cisim ve roketlerin maruz kaldıkları kuvvetleri hesaplar ve hareketlerini günceller. Zaman adımı boyunca simülasyonun ilerlemesini sağlar.

Bu yapı sayesinde, sistemdeki tüm cisim ve roketler birbirleriyle etkileşime geçerek, karmaşık yörünge hareketleri hesaplanabilir.

4 Programın Çalışma Hızı ve Bellek Yönetimi İyileştirmeleri

Programın performansını artırmak ve bellek kullanımını optimize etmek için çeşitli iyileştirmeler yapılmıştır. Bu iyileştirmeler, simülasyonun daha hızlı çalışmasını sağlarken, bellek yönetiminin verimli olmasına da katkıda bulunmuştur.

Çalışma Hızı İyileştirmeleri

- **Zaman Adımı Optimizasyonu**: Simülasyonun doğruluğu ve hızı arasında denge sağlamak için `timestep` sabiti kullanılmıştır. Küçük zaman adımları (`timestep = 0.01`) kullanılarak hesaplama doğruluğu artırılmış, ancak bu değer hesaplama süresini çok uzatmayacak şekilde seçilmiştir. Bu optimizasyon, sistemin hareket denklemlerini doğru ve hızlı bir şekilde çözmesini sağlar.
- **Vektör İşlemlerinin Optimizasyonu**: `vector` sınıfında tanımlanan toplama, çıkarma ve skaler çarpma işlemleri doğrudan matematiksel işlemlerle gerçekleştirilmiştir. Karmaşık hesaplama yapılarından kaçınılarak işlem süreleri azaltılmıştır. Bu sayede, cisimler arasındaki kuvvet hesaplamaları ve konum güncellemeleri daha hızlı yapılabilmiştir.

Bellek Yönetimi İyileştirmeleri

- **Dinamik Hafıza Kullanımı**: Cisim ve roket nesneleri, `std::vector<body*>` yapısı içinde dinamik olarak tutulmaktadır. Bu yapı, simülasyon sırasında bellek kullanımını esnek hale getirir ve gerektiğinde yeni cisimler eklenmesine olanak tanır.

- **Bellek Kaçağı Önlemleri**: Programda kullanılan tüm nesneler, otomatik ömür yönetimi sayesinde program sonlandığında temizlenmektedir. `std::vector` kullanımı, dinamik hafıza tahsisinde bellek kaçağı riskini azaltır. Ayrıca, nesnelerin kopyalanmasını ve taşınmasını doğru yöneten yapılar kullanılarak bellek verimliliği artırılmıştır.

5 Kalıtımın İncelenmesi

Programda OOP'nin temel prensiplerinden biri olan kalıtım kullanılmıştır. Kalıtım, sınıflar arasında kod tekrarını azaltmak ve esnekliği artırmak amacıyla kullanılmıştır. Projede aşağıdaki kalıtım ilişkileri mevcuttur:

- `rocket` **Sınıfı** `body` **Sınıfından Türetilmiştir**: `rocket` sınıfı, `body` sınıfının özelliklerini ve fonksiyonlarını miras alır. `rocket` sınıfı, `body` sınıfının temel hareket güncelleme fonksiyonunu genişleterek roketlere özgü itme kuvveti ve kütle azalmasını hesaplar.
  - **Amaç**: Roket hareketlerinin, noktasal cisim hareketlerinden farklı olarak itme kuvveti ve yakıt tüketimi ile modellenmesini sağlamak.
- `universe` **Sınıfı** `body` **ve** `rocket` **Nesnelerini Kullanır**: `universe` sınıfı, sistemdeki tüm cisim ve roketlerin etkileşimlerini yönetir. `body` ve `rocket` sınıflarından oluşturulan nesneler üzerinde işlem yapar.
  - **Amaç**: Tüm cisimlerin ve roketlerin hareketlerini ve etkileşimlerini ortak bir yapı altında yönetmek.

Bu kalıtım yapısı sayesinde, roket hareketi gibi özel davranışlar `body` sınıfının temel özelliklerini tekrar etmeden genişletilmiş, programın modülerliği ve genişletilebilirliği sağlanmıştır.

6 Çok Biçimlilik

Programda çok biçimlilik (polymorphism) kullanılmıştır. Çok biçimlilik, farklı sınıfların ortak bir arayüzü paylaşarak kendi özel davranışlarını tanımlamasına olanak tanır. Projede kullanılan çok biçimlilik şunlardır:

Fonksiyonun Yeniden Tanımlanması (Override)

- `update` **Fonksiyonu**: `body` sınıfında tanımlanan `update` fonksiyonu, `rocket` sınıfında yeniden tanımlanmıştır. Roketlerin hareketi, cisimlerin hareketinden farklı olduğu için `update` fonksiyonunun roketlere özgü bir versiyonu oluşturulmuştur. Bu sayede, roketlerin hareket hesaplamalarında itki kuvveti ve kütle azalması gibi ek faktörler dikkate alınır.

Sanal Fonksiyonlar (Virtual Functions)

- `update` **Fonksiyonunun Sanal Yapısı**: `body` sınıfındaki `update` fonksiyonu sanal (virtual) olarak tanımlanmıştır. Bu sayede, `body` sınıfından türetilen sınıflar (örneğin `rocket`) bu fonksiyonu kendi ihtiyaçlarına göre yeniden tanımlayabilir. Program çalışırken, hangi sınıfın nesnesi kullanılıyorsa o sınıfa ait `update` fonksiyonu çağrılır. Bu dinamik bağlama (dynamic binding) mekanizmasını sağlar.

Ortak Arayüz Kullanımı

- **Ortak Davranışlar**: `body` ve `rocket` sınıfları, aynı arayüzü paylaşarak benzer işlemler için aynı fonksiyon adlarını kullanır. Bu sayede, `universe` sınıfı gibi yapılar, `body` ve `rocket` nesneleri üzerinde aynı işlem mantığını uygulayabilir.

Bu çok biçimlilik yapıları, kodun esnekliğini ve genişletilebilirliğini artırmakta, farklı türdeki nesnelerle aynı arayüz üzerinden çalışmayı mümkün kılmaktadır.

7 Yörünge Hareketleri ve Simülasyon Görselleştirmesi

Bu bölümde, üç cisim problemi ve roket hareketlerinin simülasyon çıktısı görselleştirilmiştir. Simülasyon, cisimlerin kütle çekim kuvvetleri altında izledikleri yörüngeleri ve roketlerin itki kuvveti ile hareketlerini göstermektedir.

Açıklama:

- **Kırmızı Yörünge**: Roketin hareketini temsil eder. Roket, itki kuvvetiyle hareket ederken, kütle çekim kuvvetlerinin etkisiyle karmaşık bir yörünge izler.
- **Yeşil Yörünge**: Bir cisim tarafından izlenen yörüngeyi gösterir. Bu cisim, diğer cisimlerin çekim etkisiyle yön değiştirir.
- **Mavi Yörünge**: Diğer bir cisim tarafından izlenen yörüngeyi ifade eder. Bu cismin hareketi de diğer cisimlerin çekim kuvvetlerine bağlıdır.

Bu görsel, üç cisim problemi ve roket hareketinin karmaşık doğasını ve kütle çekim etkilerinin yörüngeler üzerindeki dinamik etkilerini net bir şekilde ortaya koymaktadır.

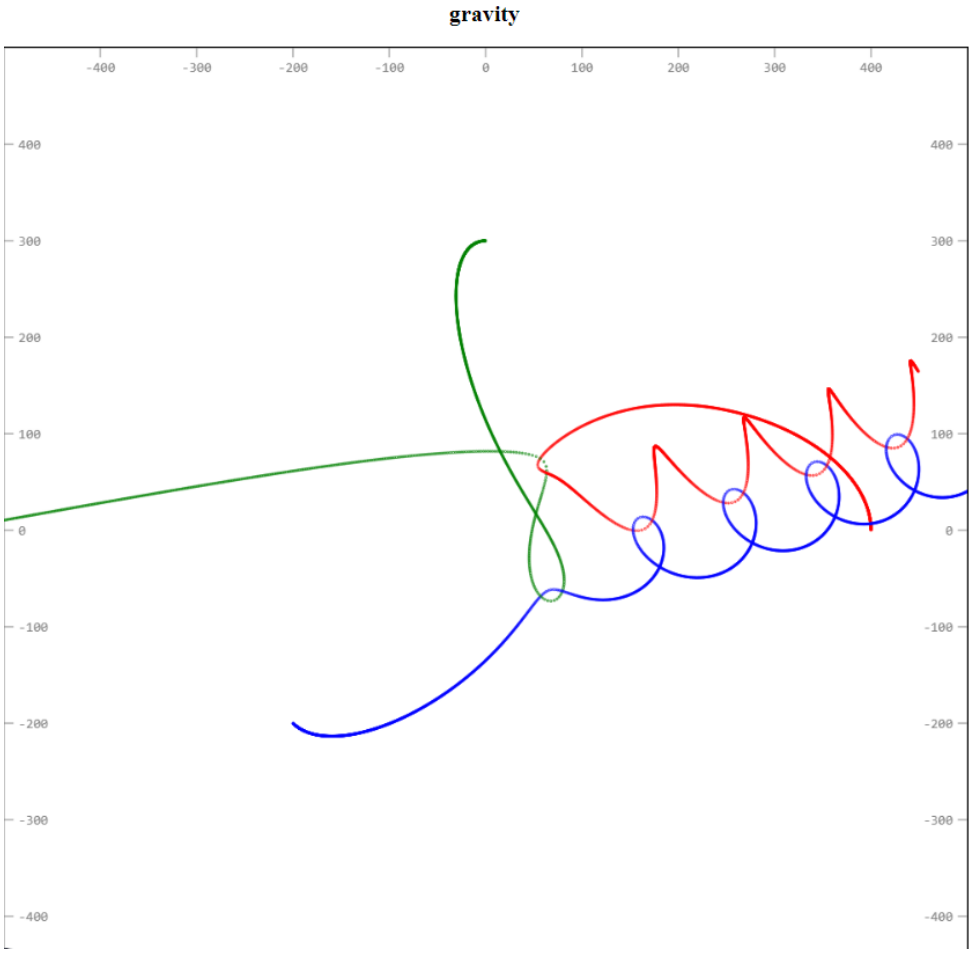

Figure 1: Üç cisim problemi ve roket hareketinin simülasyon yörüngeleri

8 Fiziksel Gözlemler

Simülasyon çıktıları üzerinden şu fiziksel gözlemler yapılabilir:

- **Ağırlık Merkezinin Hareketi**: Sistemdeki tüm cisimlerin kütlelerine ve konumlarına bağlı olarak ağırlık merkezi hesaplanır. Roketlerin kütle kaybı veya cisimlerin hareketi sonucunda ağırlık merkezinin dinamik olarak nasıl değiştiği gözlemlenebilir.
- **Momentumun Korunumu**: Sisteme dış kuvvet uygulanmadığında toplam momentum korunur. Roketlerin itki kuvveti devreye girdiğinde, sistemin toplam momentumunun roket itişine bağlı olarak nasıl değiştiği analiz edilebilir.
- **Roket İtki Kuvvetlerinin Etkisi**: Roketlerin farklı itki kuvvetleri ve yakıt tüketim oranları, yörüngelerini doğrudan etkiler. İtki kuvvetinin büyüklüğüne ve yönüne göre roketin izlediği yörünge incelenebilir.