МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ В. Н. КАРАЗІНА

Фізичний факультет
Кафедра астрономії та космічної інформатики

«Допущено до захисту»
В.о. зав. кафедри астрономії та
космічної інформатики
проф. Шкуратов Ю. Г. ____ _____
«14» червня 2024 р

Оцінка «відмінно»
Голова ДЕК Тишковець В.П.
«21» червня 2024 р.

Ільченко Денис Олександрович

Ефекти гравітаційного лінзування на кільцевій галактиці

Дипломна робота на здобуття
освітнього рівня
«Бакалавр» за спеціальністю
104 – «фізика та астрономія»
освітньо-наукова програма -
«астрономія»

_____ _____
(підпис студента)

Науковий керівник:
професор кафедри астрономії та
космічної інформатики,
доктор фіз.-мат. наук,
Баннікова Олена Юріївна

_____
(підпис керівника)

Рецензент:
Доцент кафедри теоретичної фізики
Котвицький А. Т.

_____
(підпис рецензента)

Харків 2024

Анотація

Ільченко Д.О. Ефекти гравітаційного лінзування на кільцевій галактиці

Дипломна робота на здобуття ступеня вищої освіти «бакалавр» за спеціальністю 104 «Фізика та астрономія». Харківський національний університет імені В.Н. Каразіна, Харків, 2024.

В рамках дипломної роботи досліджувались ефекти сильного гравітаційного лінзування на кільцевих системах. Було отримано рівняння критичних ліній та каустик для системи однорідного диску з центральною масою. Для системи однорідного тора з центральною масою було отримано нові рівняння лінзи у елементраних функціях. Для цієї системи виявлено новий ефект - утворення двох кілець Ейнштейна у області на торі. Для обох систем побудовано графіки, для відображення зміщення та підсилення зображення. Було створено програмне забезпечення для візуалізації та створення анімацій ефектів гравітаційного лінзування.

Abstract

Ilchenko D. A. Effects of gravitational lensing on a ring galaxy

Thesis for the degree of Higher Education «bachelor» in specialty 104 «Physics and astronomy». V. N. Karazin Kharkiv National University, Kharkiv, 2024.

The effects of strong gravitational lensing on ring systems were investigated in the thesis. The equations of critical lines and caustics for the system of a homogeneous disk with a central mass were obtained. For the system of a homogeneous torus with a central mass, new lensing equations in elementary functions were obtained. For this system, a new effect is found - the formation of two Einstein rings in a region on the torus. For both systems, graphs were constructed to show the shift and magnification of the image. Software was created to visualize and create animations of gravitational lensing effects.

Зміст

Вступ **4**

1 Історія розвитку та спостереження гравітаційних лінз **6**

1.1 Відхилення світла точковою лінзою 6

1.2 Рівняння гравітаційної лінзи. Кільце Ейнштейна 7

1.3 Спостережні данні 12

2 Моделювання ефектів гравітаційного лінзування **17**

2.1 Численный метод трасування променів 17

2.2 Реализация алгоритма трасування променів на C++ 18

2.3 Візуалізація з допомогою OpenGL 19

2.4 Тестування коду на класичних лінзах 21

2.5 Коефіцієнт підсилення 23

3 Осесиметричні системи **26**

3.1 Однорідний диск 26

3.2 Розрахунок рівняння тора в термінах R_{tor} і r_0 34

Висновки **42**

Додаток А: Зображення та графіки гравітаційного лінзування для різних систем **43**

Додаток В: Код програми для візуалізації гравітаційного лінзування **58**

Список використаних джерел **86**

Вступ

На сьогоднішній день вивчення ефектів гравітаційного лінзування є важливою задачою. Завдяки космічним телескопам, таких як Hubble Space Telescope і новому James Webb Space Telescope був отриман великий масив спостережних данних. Аналіз цих зображень є потужним методом визначення розподілу баріонної та темної матерії у просторі.

Актуальність цієї роботи полягає в дослідженні ефектів гравітаційного лінзування на тороїдальних систмах, що є поширеними у всесвіті. Прикладом таких систем є кільцеві галактики типу об'єкта Хога, або затінюючи тори у активних ядрах галактики (АЯГ), що представляють великий інтерес для астрофізики.

Грунтуючись на цій темі були поставлені наступні задачі:

1. Створення програмного забезпечення для візуалізації гравітаційного лінзування.

2. Освоїти теорію гравітаційного лінзування на точковій лінзі.

3. Знайти залежність кількості зображень від положення джерела для системи однорідного кільцевого диску з центральною масою.

4. Розглянути нову систему - кільцевий диск з розподілом густини що відповідає однорідному тору з центральною масою.

Новизна цієї роботи полягає в досліджені нової, більш складної гравітациінолінзової системи, що є кращим наближенням до реальних астрофизичних об'єктів.

Практичне значення роботи полягає у освоєнні теорії гравітаційного лінзування та отримання фізичних характеристик для обраних систем.

Наукове значення отриманих результатів полягає в тому, що для системи однорідний тор з центральною масою були знайдені нові ефекти, що не спостерігаються у раніше розглянутій системі однорідного диску з центральною

масою. Розроблене програмне забезпечення в майбутньому може слугувати для створення датасету для машиного навання для знаходження описаних ефектів на реальних знімках.

1 Історія розвитку та спостереження гравітаційних лінз

1.1 Відхилення світла точковою лінзою

Природа світла довгий час залишалась для людей загадкою і питання чи взаємодіє воно гравітаційно ймовірно ставилось давно. У 1796 році Генрі Кавендіш вивів формулу для кута відхилення світла від прямої лінії при проходжені повз масивне тіло, але не опублікував її. Тож вперше вона була освітлена у роботі Іоганна Георга фон Зольднера 1801 року [8], де кут відхидення променя світла має вигляд:

$$\alpha = \frac{2GM}{c^2\xi} \tag{1}$$

де G – гравітаційна стала, M – маса гравітуючого тіла, c - швидкість світла, ξ – прицільний параметр. Проте вже на початку 19го сторічя експерементально були підтверджені хвильові властивості світла і питання викривлення променя світла масивним тілом ще довго не розглядалось.

У 1907 році Альберт Ейнштейн сформулював принцип еквівалентності згідно з яким гравітаційна маса не відрізняється від інерційної. А одже рух у гравітаційному полі не можливо відрізнити від руху з прискоренням. Також постулювавши, що швидкість світла в усіх інерційних системах стала, що можливо дослідити в уявному експерименті Ейнштейна. Ліфт що падає у гравітаційному полі має дві скрізні шпарини через які проходить світло. У системі відліку ліфта він знаходиться у стані спокою а одже світло проходть по прямій крізь шпарини. Тоді в силу принципу еквівалентості в системі відліку Землі щоб світло пройшло через обидві шпарини воно мало б відхилитись від прямої лінії - одже гравітація впливає на рух світла.

У 1911 року Ейнштейн провів розрахунок кута відхилення у рамках спеціальної теорії відносності і він збігся з результатами Кавендіша і Зольднера.

Ейнштейн звернувся у листі до сера Джорджа Еллері Хейла, який запропонував виміряти кут відхилення зір від істинного положення Сонцем під час затемнення. Перші спроби були проведені в 1914 році у Феодосії, проте спостереженням завадила Перша світова війна. У 1916 році Ейнштейн повторив розрахунок у рамках загальної теорії відносності (ЗТВ) і отримав кут у двічі більше.

$$\alpha = \frac{4GM}{c^2\xi} = \frac{2r_g}{\xi} \tag{2}$$

де $r_g = 2GM/c^2$ - гравітаційний радіус лінзи. Підставивши у формули (1) і (2) масу Сонця $M = M_\odot = 1.99 \times 10^{30}$ кг і прирівнявши прицільний параметр до його радіуса $\xi = R_\odot = 6.96 \times 10^8$ м отримаємо кут відхилення 0".875 і 1".75 відповідно. Тож спостереження відхилення світла могло стати підтвердженням ЗТВ.

У 1918 року Вільям Воллес Кемпбелл провів спостереження у Вашингтоні але через низьку точність відхилення не було помічено, що викликало сумніви у ЗТВ. Проте вже у 1919 році провели ще дві експедиції на острові Принсіпі та на півночі Бразилії організовані Артуром Еддінгтоном. Вони спостерігали затемнення на фоні Гіад, що завдяки великій кількості зір у скупчені дозволило усунути випадкові похибки. Вимірюваннння показали кут відхилення 1.60 ± 0.31 та 1.98 ± 0.12 секунди дуги, тож відхилення світла на цей кут є одним із тестів ЗТВ, що підтверджують її правильність.

1.2 Рівняння гравітаційної лінзи. Кільце Ейнштейна

Тепер знайдемо залежність координат зображень від координат джерела. Розглянемо промінь світла, що йде від джерела. На Рис. 1 зображений двовимірний випадок. Знайдемо залежність ξ від координати джерела η.

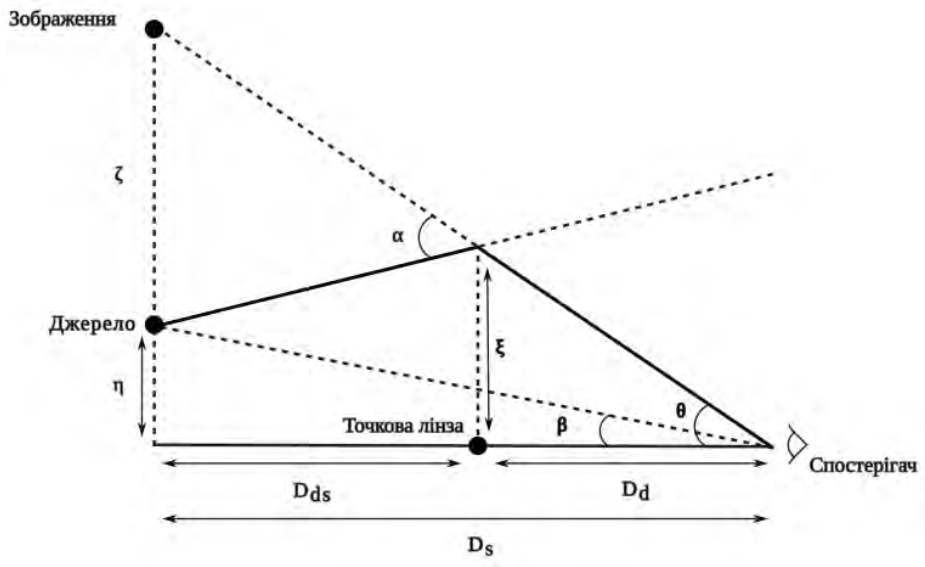


Рис. 1: Схема руху проміня від джерела до спостерігача

З подібності двох трикутників отримаємо рівність:

$$\frac{\xi}{D_d} = \frac{\zeta + \eta}{D_s}.$$

з урахуванням дальності джерела $\alpha \to 0 \Rightarrow \zeta = D_{ds}\alpha = 2D_{ds}\frac{r_g}{\xi}$

$$\frac{\xi}{D_d} = \frac{\eta + D_{ds}\frac{2r_g}{\xi}}{D_s} \Rightarrow \frac{\eta + 2r_g D_{ds}}{\xi} = \xi\frac{D_s}{D_d}$$

Остаточно рівняння гравітаційної лінзи має вигляд:

$$\eta = \frac{D_s}{D_d}\xi - \frac{2r_g D_{ds}}{\xi}. \tag{3}$$

У випадку коли джерело, лінза і спостерігач лежать на одній прямій $\eta = 0$ розв'язок рівняння має вид

$$\xi_0 = \pm\sqrt{2r_g\frac{D_d Dds}{D_s}}. \tag{4}$$

В такому випадку світло від джерела огинає лінзу з усіх сторін Рис. 2 і спостерігач буде бачити кільцеподібне зображення - кільце Ейштейна, радіус

якого визначається формулою (4).

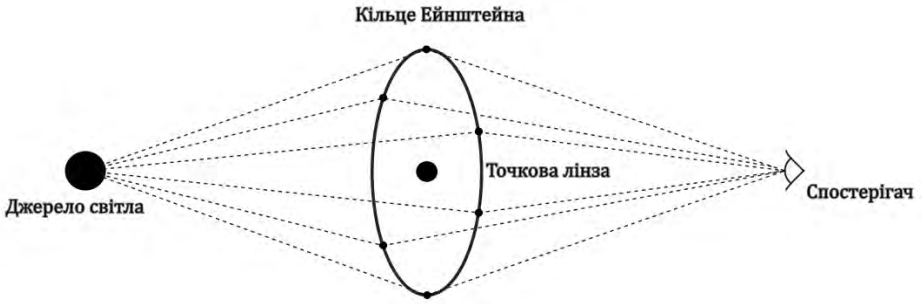


Рис. 2: Схема формування кільца Ейнштейна.

Цей рузультат був отриманий Ейнштейном у 1936 році [6].Однак оцінка ефекту при лінзуванні зорі на зорі була настільки маленькою, що навіть сам Ейнштейн розглядав його скоріше як цікавий з точки зору теорії без можливості його спостерігати. Ефект виявляється значним якщо розглядати лінзування галактики на галактиці, оскільки маса галактик сутєво більша, а відстань джерело-лінза, лінза-спостерігач можуть бути одного порядку, що і збільшує значення радіуса кільця Ейнштейна, згідно формули (4). Крім того прогрес у розвитку спостережних засобів дозволив досягти високої роздільної здатності далеких об'єктів, що також дало імпульс до розвитку напрямку дослідження астрофізичних об'єктів методами гравітаційного лінзування. Ми переконаємось пізніше, що при лінзуванні далеких джерел на галактиках утворення кілець значно ймовірніше і вже є цілий список таких зображень, а з зорями пов'язані ефекти мікролінзування.

Рівняння лінзи для випадку точкової лінзи можна узагальнити на двовимірний випадок. Дійсно, в більш загальному випадку джерело визначається двома координатами на площині джерела, тобто, $\vec{\eta} = (\eta_1, \eta_2)$. Відповідно координати прицільного параметру $\vec{\xi} = (\xi_1, \xi_2)$ (Рис. 3)

Таким чином рівняння гравітаційної лінзи у векторному вигляді має вид:

$$\vec{\eta} = \frac{D_s}{D_d}\vec{\xi} - \frac{2r_g D_{ds}}{\xi^2}\vec{\xi}. \tag{5}$$

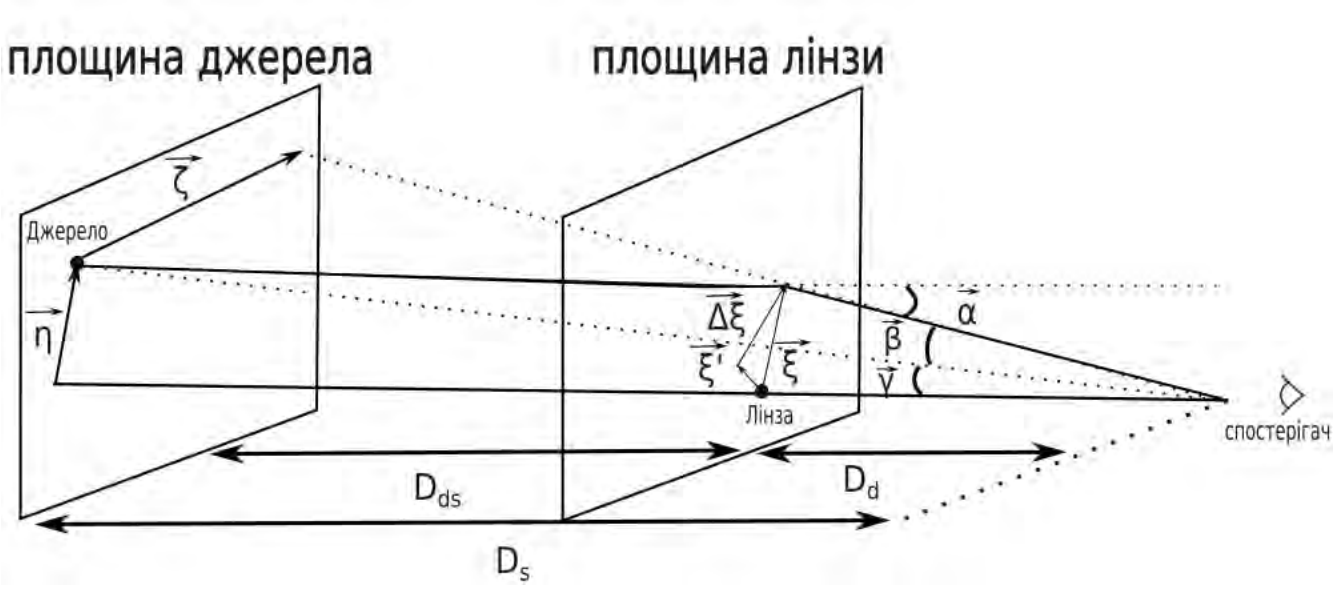


Рис. 3: 3D схема викривлення променя світла і положення зображення

Нормуємо рівняння (5) на радіус Ейнштейна $\vec{x} = \frac{\vec{\xi}}{\xi_0}$

$$\frac{\vec{\eta}}{\xi_0} = \frac{D_s}{D_d}\vec{x} - \frac{2r_g D_{ds}}{\xi^2}\vec{x} \Rightarrow \frac{D_d}{D_s}\frac{\vec{\eta}}{\xi_0} = \vec{x} - \frac{2r_g D_{ds} D_d}{D_s \xi^2}\vec{x}$$

Введемо ще одне нормування $\vec{y} = \frac{D_d}{D_s}\frac{\vec{\eta}}{\xi_0}$

$$\xi_0^2 = 2r_g \frac{D_d D_{ds}}{D_s} \Rightarrow \frac{2r_g D_{ds} D_d}{D_s \xi_0^2}\frac{\xi_0^2}{\xi^2} = \frac{\xi_0^2}{\xi^2} = \frac{1}{x^2}$$

Остаточно отримуємо:

$$\vec{y} = \vec{x} - \frac{\vec{x}}{x^2} \tag{6}$$

Рівняння (6) є обезрозміреною формулою лінзи де $\vec{y}$ - нормовані координати джерела, а $\vec{x}$ - нормовані координати зображення.

Основною метою є знаходження координат зображень, що легко можна зробити для рівняння точкової лінзи. Рішення рівняння (6) має вигляд:

$$\vec{x}_{\pm} = \vec{y}\left(\frac{1}{2} \pm \sqrt{\frac{1}{4} + \frac{1}{|y|^2}}\right). \tag{7}$$

У скалярній формі рівняння (6) і його рішення наступні:

$$y = x - \frac{1}{x} \Rightarrow x^2 - xy - 1 = 0 \Rightarrow x_{\pm} = \frac{y \pm \sqrt{y^2+4}}{2}$$

Попередні рівняння були отримані для випадку точкової лінзи. Для більш загального випадку лінзи, що складається з набору точкових лінз рівняння (6) прийме вид:

$$\vec{y} = \vec{x} - \vec{\hat{\alpha}}\,, \tag{8}$$

де $\vec{\hat{\alpha}} = \frac{D_{ds}D_d}{D_s}\frac{\vec{\alpha}}{\xi_0}$ нормований кут відхилення, тож маємо рівняння лінзи через кут відхилення. Кут відхилення буде визначатись як векторна сума кутів відхилення, що створює кожна точкова лінза з масою m_i, нормовану на повну масу системи, окремо:

$$\vec{\hat{\alpha}} = \frac{D_{ds}D_d}{D_s\xi_0}\frac{4G}{c^2\xi_0}\sum_{i=1}^{N} m_i \frac{\vec{x} - \vec{x_i}}{|\vec{x} - \vec{x_i}|^2}\,. \tag{9}$$

Для випадку неперервного розподілу маси перейдемо від сумування до інтегрування:

$$\vec{\hat{\alpha}} = \frac{D_{ds}D_d}{D_s\xi_0}\frac{4G}{c^2\xi_0}\int \frac{\vec{x} - \vec{x'}}{|\vec{x} - \vec{x'}|^2}\,dm\,. \tag{10}$$

Рівняння лінзи (8) з врахуваннм (9) або (10) є основним рівнянням теорії гравітаційного лінзування, що дозволяє отримати зображення джерела для довільного розподілу густини в лінзі. Це рівняння в загальному випадку неможливо вирішити, оскільки координати зображень входять туди складним чином через інтеграл і вільний доданок. Також не завжди вдається отримати саме рівняння лінзи в елементарних функціях і тільки для особливих випадків розподілу густини в лінзі це можливо зробити(як, наприклад, для точкової лінзи). У розділі 3.2 ми побачимо, що для запропонованого в роботі розподілу лінзи, який відповідає розподілу речовини в торі, рівняння можливо записати в елементарних функціях.

Однак навіть для знаходження розв'язку потрібні спеціальні методи, оскільки рівняння лінзи не розв'язується щодо координат зображення. Тому використовується метод трасування променів, який буде розглянуто в розділі 2.1.

1.3 Спостережні данні

Легше всього знаходити лінзи які створюють зображення квазарів, або кластерів галактик. Відстань між двома зображеннями при лінзуванні на масивній галактиці порядку 1 арксекунди. Також для спостереження гравітаційного лінзування корисно розглядати компактні радіо джерела. Так як лінзування наближено не змінює поверхневої яскравості і спектр. Критеріями для зображень від точкового джерела є:

1. Знаходження кількох об'єктів на малій відстані.
2. Однакові спектри(хоча насправді спектри можуть дещо відрізнятися через те як промені проходять крізь лінзу).
3. Однакові червоні зміщеннями.
4. Відношення потоків схожі для різних зображень(також може дещо змінюватись через лінзування, та через те що світло надходить з різних частин джерела).
5. Поруч має бути кандидат на гравітаційну лінзу з червоним зміщенням менше ніж у зображення.
6. Часові варіації на різних зображеннях, які корелюють між собою.

При лінзуванні протяжного об'єкту цілком достатньою умовою є наявність сильно витягнутих чи кільцеподібних зображень. Також дуже корисною є наявність наднової, що допомагає оцінити криву блиску та часові варіації.

Загалом зображення від протяжних об'єктів легше реєструвати, проте їх і складніше обробляти. Першою знайденою гравітаціною лінзою була система QSO 0957 + 561 представляє з себе два зображення квазарів з подібними спектрами і червоними зміщеннями, Рис. 4. А перші гравітаційні дуги спостерігались у системі Abell 370, Рис. 5.

Рис. 4: Зобреження системи QSO 0957 + 561 отримане HST [1]

На сьогоднішній день зображення гравітаційного лінзування отримувало багато інструментів. Серед них Hubble Space Telescope (HST), James Webb Space Telescope (JWST), Atacama Large Millimeter/submillimeter Array (ALMA), Sloan Digital Sky Survey (SDSS) та інші. Ці інстументи вже знайшли багато прикладів утворення кілець Ейнштейна приведених на Рис. 6 – 9.

У 2007 році HST знайшов два концентричних кільця Ейнштейна у області SDSSJ0946+1006 зображених на Рис.10 [7]. Як ми побачимо у розділі 3 декілька кілець Ейнштейна може утворюватись при лінзуванні на кільцевих системах.

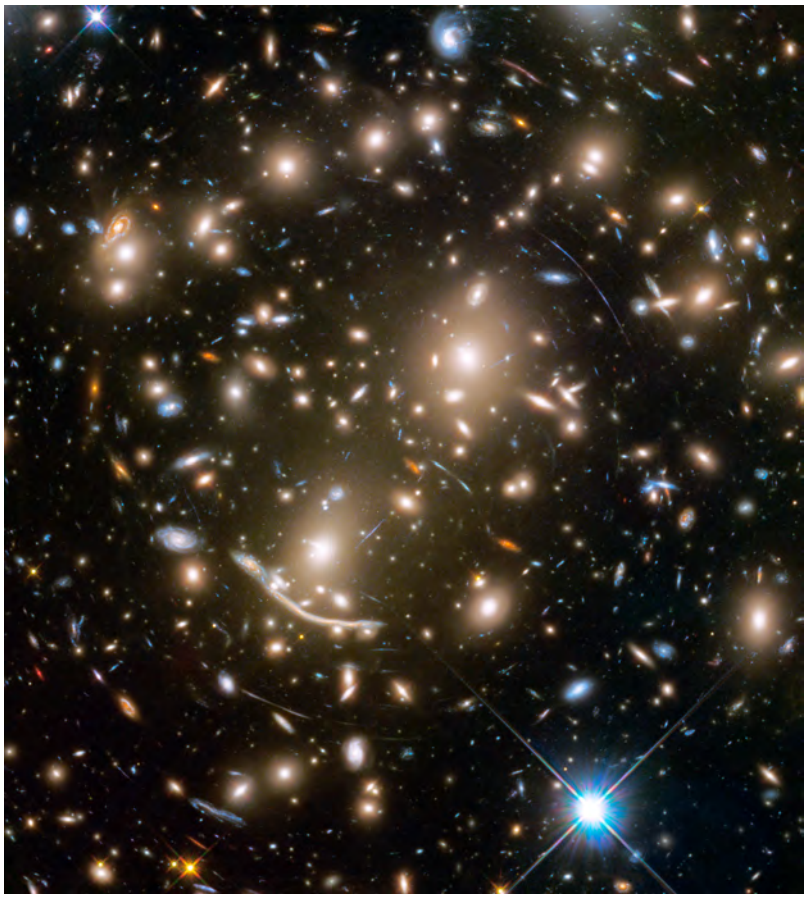


Рис. 5: Зображення системи Abell 370 отримане HST [1]

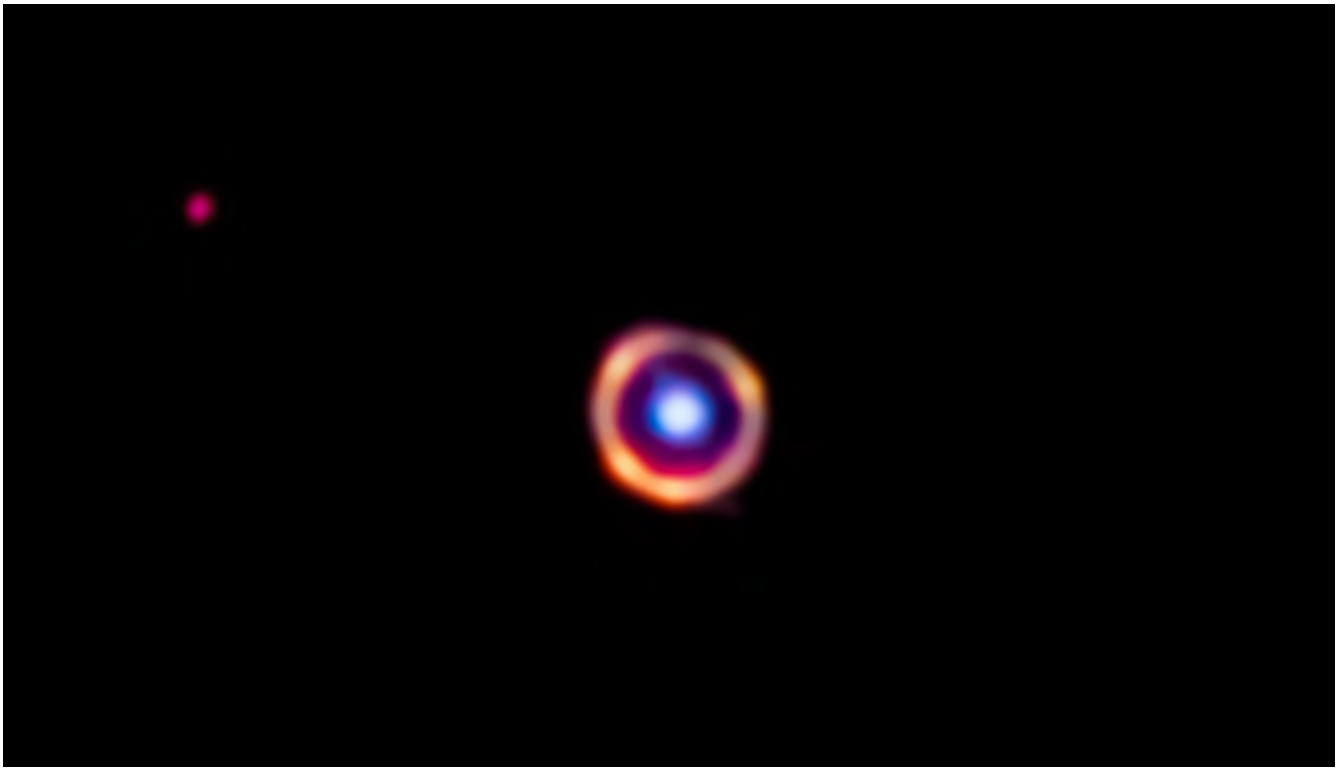
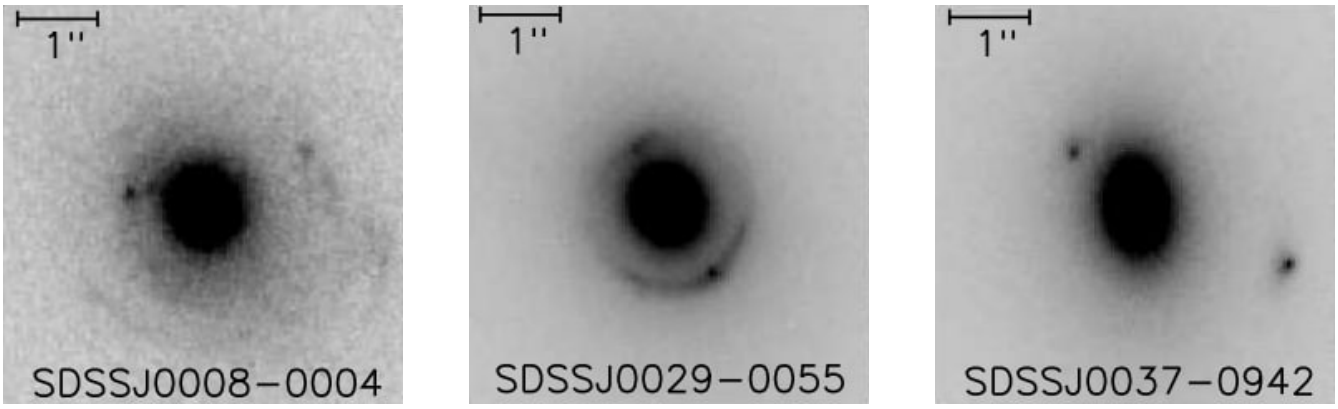


Рис. 6: Повне кільце Ейнштейна отримане JWST [2]



а б в

Рис. 7: Зображення гравітаційних лінз отримані SDSS [3]

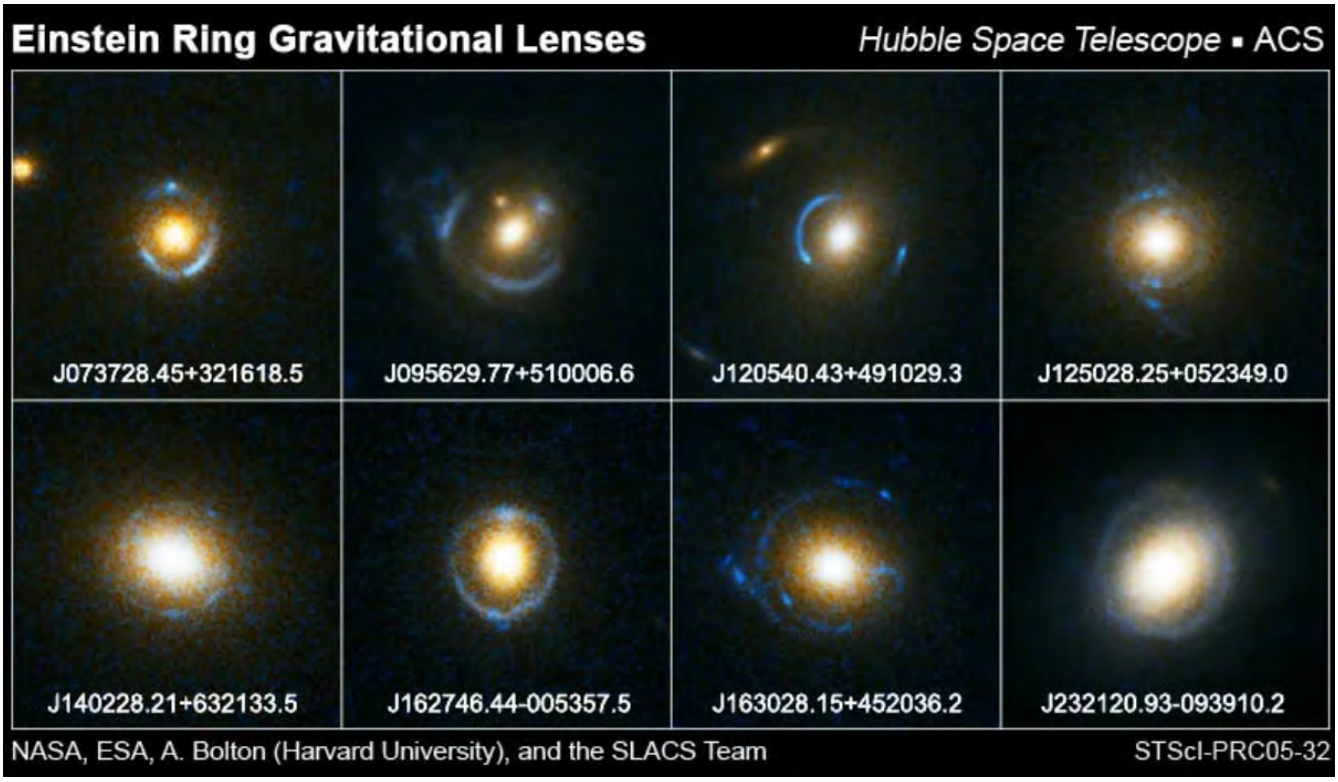


Рис. 8: Приклади кілець Ейнштейна знайдених HST [1]

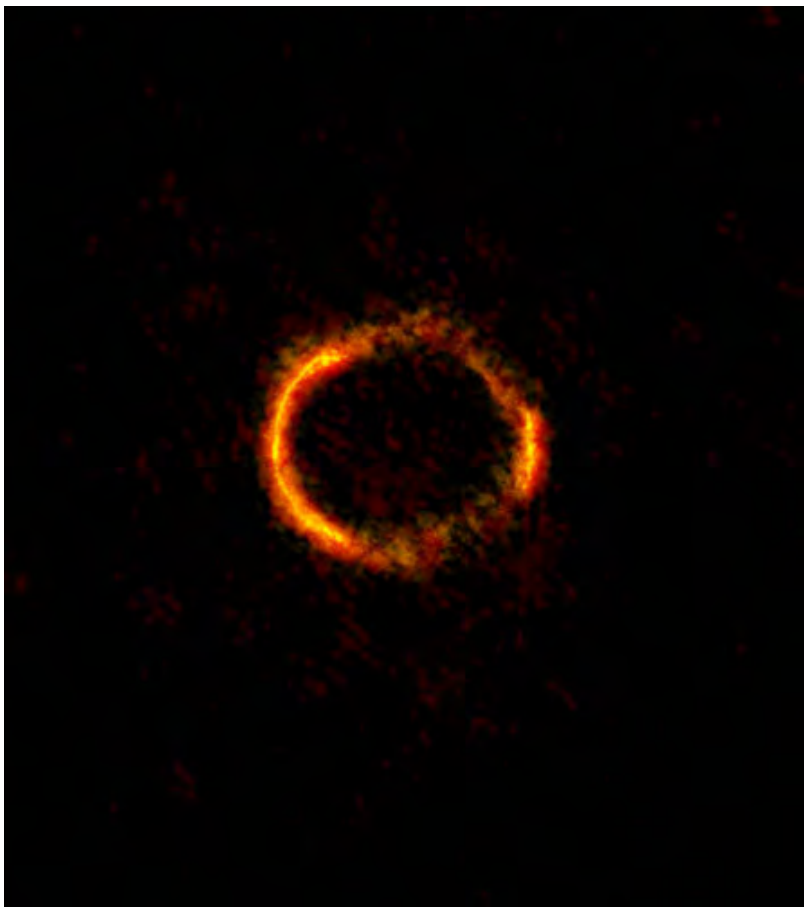
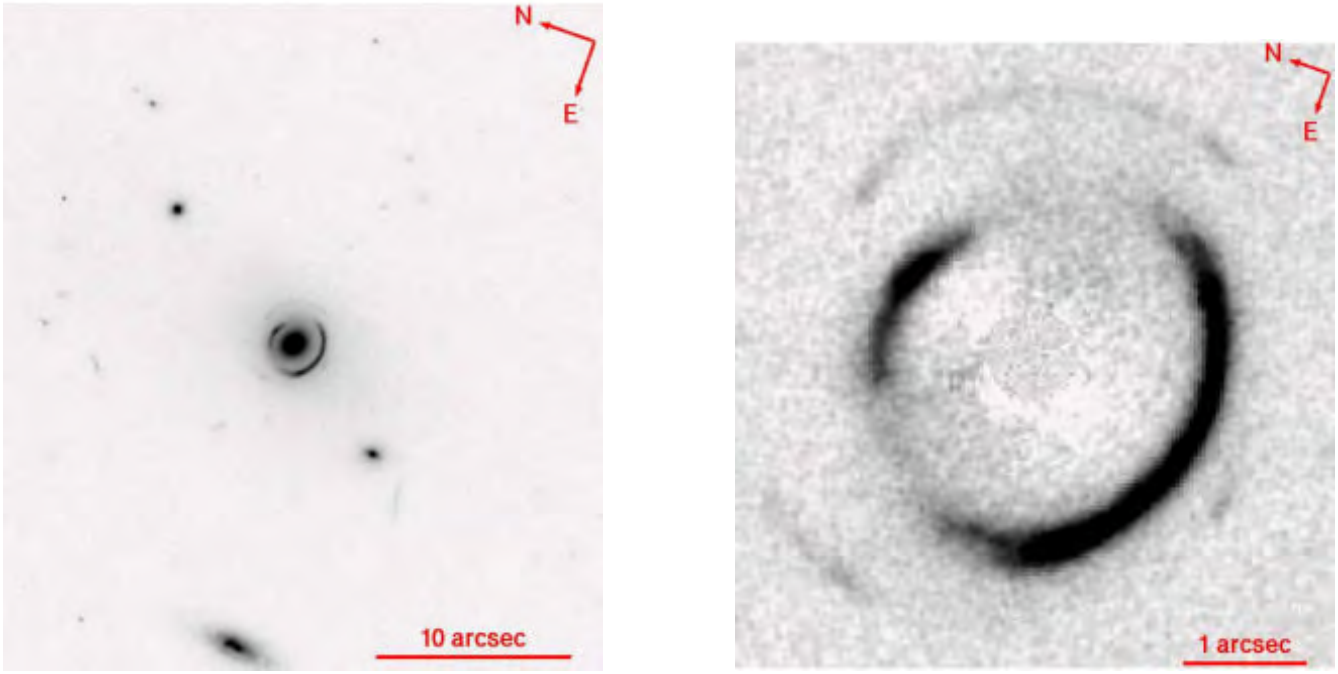


Рис. 9: Зображення гравітаційно лінзованої галактики SDP.81, отримане ALMA [4].



а б

Рис. 10: Зображення двох кілець Ейнштейна

2 Моделювання ефектів гравітаційного лінзування

2.1 Численный метод трасування променів

Метод тросування променів дозволяє за заданими координатами джерела використовуючи рівняння лінзи знайти координати зображень. У своїй роботі я використовував два підходи в залежності від того, якого виду рівняння лінзи.

Перший для випадку коли вдається отримати обернене рівняння лінзи, тобто залежність координаті зображення від координати джерела $\vec{x}(\vec{y})$. Тоді я використовую наступний алгоритм:

1. Задаю координати, розміри, колір та розподіл яскравості джерел світла.
2. Область джерела світла заповнюю точками.
3. Для кожної точки використовуючи обернені рівняння лінзи отримую відповідні координати у площинні лінзи, ці точки і утворюють зображення.
4. Нові точки отримуюють яскравість та колір початкових відповідних точок.

Другий алгоритм я використовую у випадку коли обернені рівняння лінзи отримати не вдається. Він реалізується наступними кроками:

1. Задаю координати, розміри,колір та розподіл яскравості джерел світла.
2. Заповнюю область у площині лінзи точками.
3. Для кожної точки використовуючи рівняння лінзи отримую відповідні точки у площині джерела світла.
4. Перевіряємо чи потрапила отримана точка у область джерела світла.
5. У випадку коли потрапляє, то для неї розраховується колір та яскравість, а початкова точка утворює зображення та отримує ту саму яскрвість та колір.

2.2 Реализация алгоритма трасування променів на C++

У свої моделі я створив клас Sourse_Array, який є двовимірним масивом об'єктів його вкладеного класу Sourse.

```
class Sourse_Array {
public:
    class Sourse{...};
    Sourse** sourse;
    Sourse** image;
    int n;
    int lens_type;

    Sourse_Array(int lens_type, int n){...}
    ~Sourse_Array(){...}

    void Show_Sourse(){...}
    void Show_Images(){...}
    void Show_Images_Sourse(){...}
    ...
};
```

Змінна n - відповідає за кількість джерел у масиві, а lens_type свідчить про те на якій системі відбувається лінзування (точкова лінза, точкова лінза з кільцем, тощо). Масиви sourse та image містять інформацію про джерела та зображення відповідно. методи Show використовуються для виводу на екран відповідних елементів.

Сам алгоритм тросування реалізований у класі Sourse

```
class Sourse{
public:
```

```
    double** point;
    int n;

    Sourse(){...}
    Sourse(double x, double y){...}
    ~Sourse(){...}

    void Central_Mass(const Sourse& other){...}
    void Disk_With_Central_Mass(const Sourse& other){...}
};
```

Кожне джерело складається з окремих точок. Двовимірний масив point містить інфориацію про координати та яскравість кожної такої точки. Змінна n відповідає за кількість таких точок. В методах цього класу знаходять координати зображень для різних систем лінз.

Повний код програми приведено у додатку В.

2.3 Візуалізація з допомогою OpenGL

Відображення результатів моделювання в залежності від пакету може займати багато часу, що стає серйозною проблемою при роботі із великою кількістю джерел і створенні анімацій. Я проводив моделювання за допомогою пакету Wolfram Mathematica, бібліотеки matplotlib для Python і бібліотеки OpenGL на C++. Моделювання масиву джерел 5 на 5 кожне з яких складається з 10000 точок займає 1хв 58с, 15с та 2с відповідно. Результати моделювання приведені на Рис. 11.

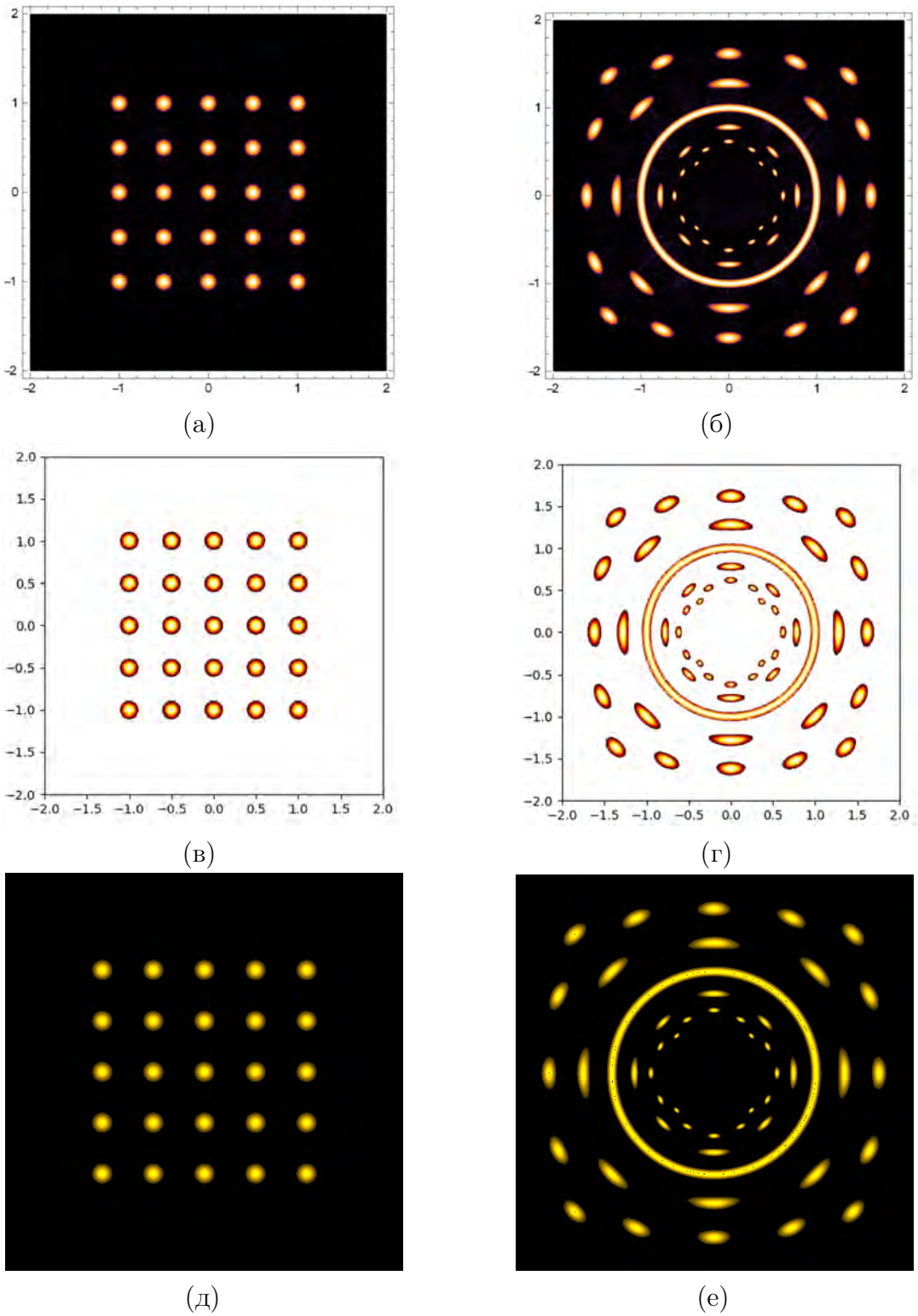


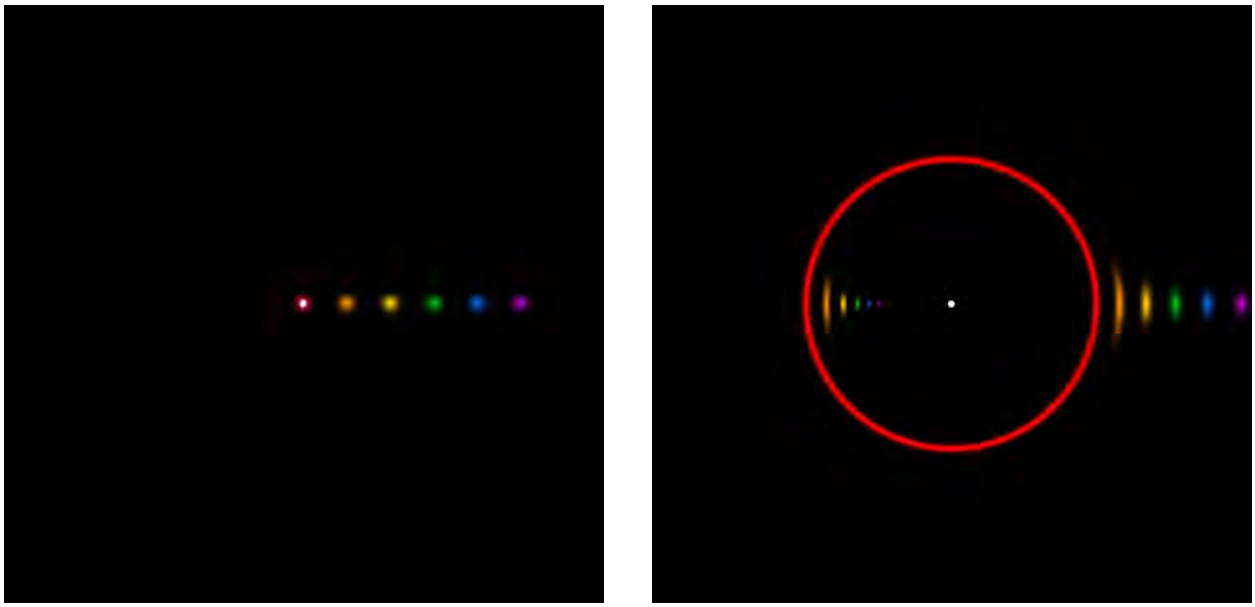
Рис. 11: (а) та (б) решітка гаусових джерел світла та відповідні зображення візуалізовані за допомогою Wolfram Matematica, (в) та (г) решітка гаусових джерел світла та відповідні зображення візуалізовані за допомогою бібліотеки matplotlib, (д) та (е) решітка гаусових джерел світла та відповідні зображення візуалізовані за допомогою, бібліотеки OpenGl

Кожна з трьох бібліотек добре візуалізує ефекти гравітаційного лінзування,

проте я зупинився на використанні OpenGl, через менший час розрахунку і можливості гнучкіше вносити необхідні зміни в прооограму. В майбутньому я планую скоротити час і підвищити якість візуалізації завдяки використанню шейдерів та розпаралелюванню розрахунків для обчислення на графічному процесорі з використанням технології CUDA.

2.4 Тестування коду на класичних лінзах

У цьому розділі ми розглянемо деякі зображення, що були отримані за допомогою моєї програми. Візуалізація відбувається за допомогою бібліотеки OpenGL. Для прикладу розглянемо рядок гаусових джерел світла. Тут і в усіх наступних зображеннях якщо не сказано іншого, радіус джерела дорівнює 0.1. Нижче приведоно отримані зображеня при лінзувані на точковій масі.

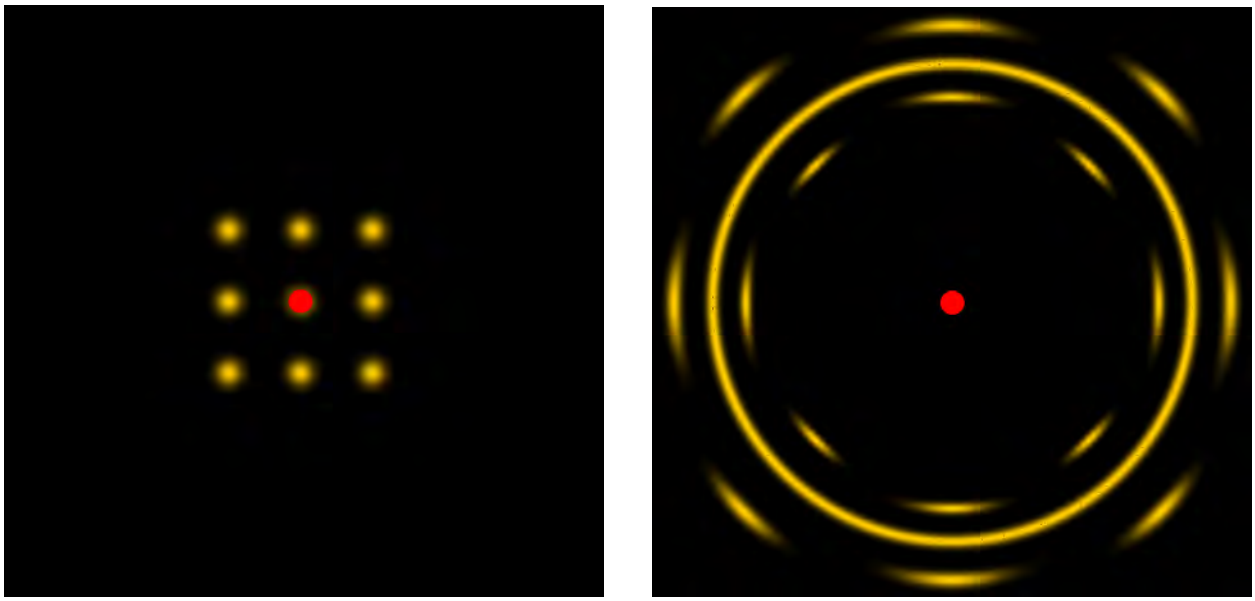


(а) (б)

Рис. 12: Лінзування кольорових зображень на точковій лінзі

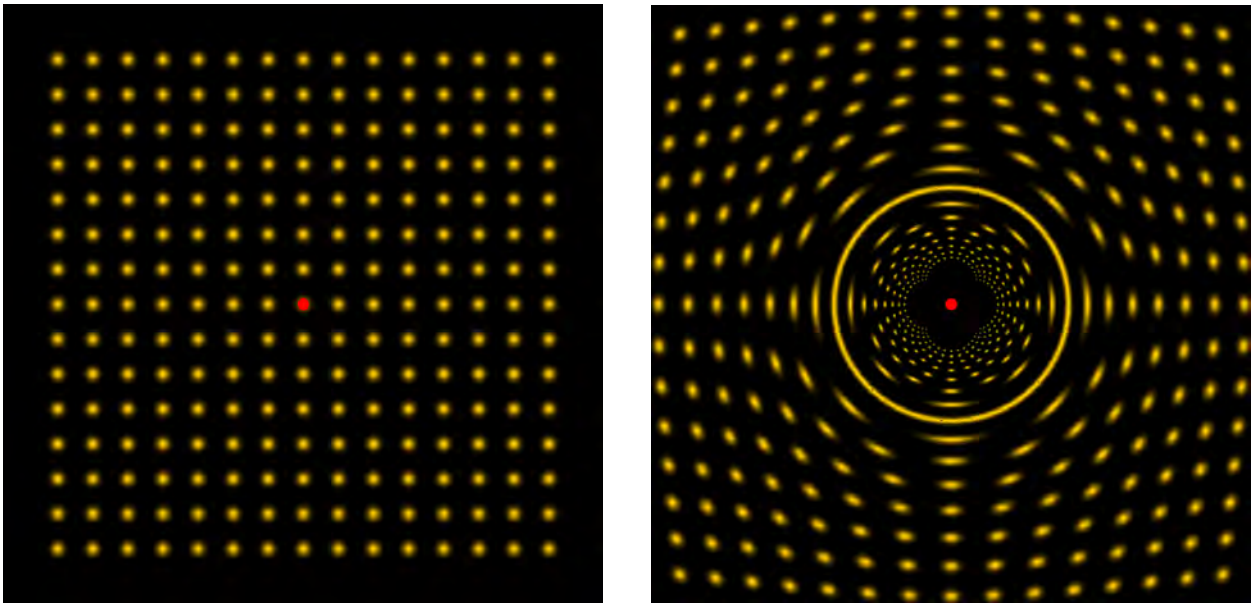
Центральне джерело створює кільце Ейнштейна, в той час як інші мають по два зображення, що мають ніби відзеркалену форму, що буде детальніше розглянуто у наступному розділі. Рисунки виконані в однаковому масштабі, тож

можемо помітити, що лінза розширює решітку, і зображення покриває більшу площу, що також розглянемо в наступному розділі. Далі розглянемо лінзування квадратної решітки гаусових джерел світла на точковій лінзі.



(а) (б)

Рис. 13: (а) решітка гаусових джерел світла 3 на 3, (б) зображення, що утворилось при лінзуванні на точковій лінзі



(а) (б)

Рис. 14: (а) решітка гаусових джерел світла 15 на 15, (б) зображення, що утворилось при лінзуванні на точковій лінзі

Бачимо, що зображення джерел світла, що знаходяться далі від гравітаціної

лінзи зазнають значно слабших змін.

2.5 Коефіцієнт підсилення

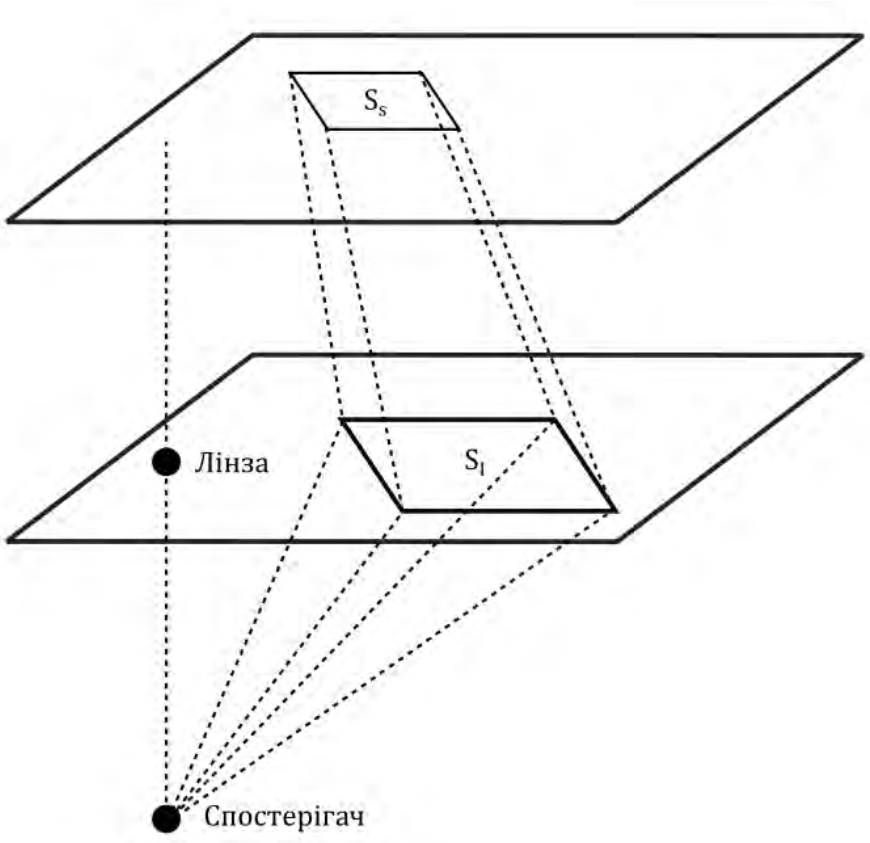


Рис. 15: Схеиа підсилення зображенння при гравітаційному лінзуванні

Гравітаційне лінзування не змінює поверхневрї яскравості джерела, тому коефіцієнт підсилення нескінченно малого джерела світла визначається як:

$$\mu = \frac{\Delta\omega}{(\Delta\omega)_0} = \left|\det \frac{\partial\beta}{\partial\theta}\right|^{-1} = \left|\det \frac{\partial y}{\partial x}\right|^{-1} = |\det A|^{-1} \tag{11}$$

Якщо детермінант має додатній(від'ємний знак), то зображення має додатню (від'ємну) парність. При від'ємній парності зобреження перевертається, а при додатній ні. Ці області розділяє критична крива де детермінант матриці рівний нулю, у площині джерела вона зветься каустикою.

Зображення з позитивною і негативною парністю з'являються і зникають разом [9]. Тому кількість зображень змінюється на два якщо джерело перете-

нає каустику. Для протяжних об'єктів коли джерело світла претинає каустику зображення з позитивною і негативною парністю зливаються. Для точкової лінзи

$$\left|\det \frac{\partial y}{\partial x}\right| = 1 - \frac{1}{x^4} \tag{12}$$

Графік залежності $\det A$ від координати зображення для точкової маси приведено на Рис. 16.

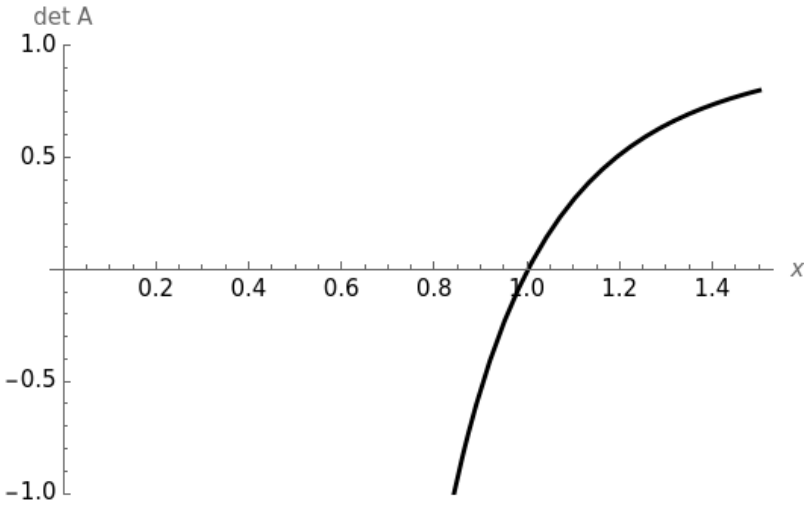


Рис. 16: Залежність детермінанта якобіана перетворення від координати зображення.

Бачимо, що для точковиої маси критичною кривою є коло з радіусом 1, тобто проходить по кільцю Ейнштейна. В середені цього кола з'являються зображення з від'ємною парністю, а зовні з додатньою. Тому на Рис. 12 ми бачимо що для одного джерела утворюються два ніби відзеркалених зображення. Підставляючи в рівняння лінзи $x = 1$ отримаємо, що каустикою є точка $y = 0$. Якщо джерело проходить через точку 0 утворюється кільце Ейнштейна, а коли джерело проходить повз утворюється дві дуги з різними знаками парності.

Далі розглянемо коефіцієнт підсилення. Використовуюючи формули (11) та (12) отримаємо, що коефіцієнт підсилення точкової лінзи визначається формулою (13)

$$\mu_{\pm} = \frac{1}{4}\left[\frac{\tilde{\beta}}{\sqrt{\tilde{\beta}^2+4}} + \frac{\sqrt{\tilde{\beta}^2+4}}{\tilde{\beta}} \pm 2\right], \tag{13}$$

де μ_+ коефіцієнт підсилення зображення з позитивною парністю,а μ_- для зображення з негативною парністю. Повне підсилення визначається як сума коефіцієнтів підсилень усіх зображень

$$\mu = \mu_+ + \mu_- \tag{14}$$

Графік коефіцієнта підсилення для точкової лінзи приведено на Рис. 17.

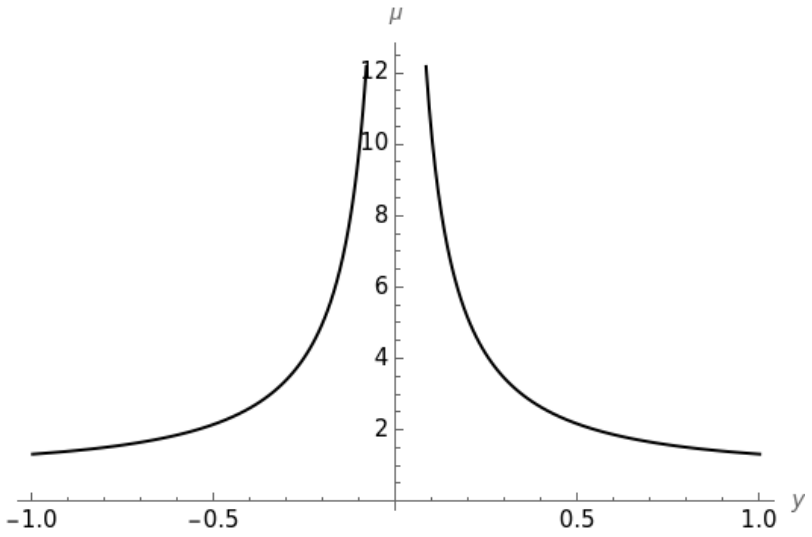


Рис. 17: Залежність коефіцієнта підсилення від координати джерела для точкової лінзи

Також важливо зазначити наступне [9]:

Теорема підсилення 1. *Серед зображень від точкового джерела, що створює прозора лінза завжди буде зображення таке саме або яскравіше за джерело без лінзи* $\mu \geq 1$

3 Осесиметричні системи

3.1 Однорідний диск

Розглянемо однорідний диск з внутрішнім радіусом r_1 та зовнішнім радіусом r_2, що нормовані на радіус кільця Ейнштейна, що зображений на Рис. 18.

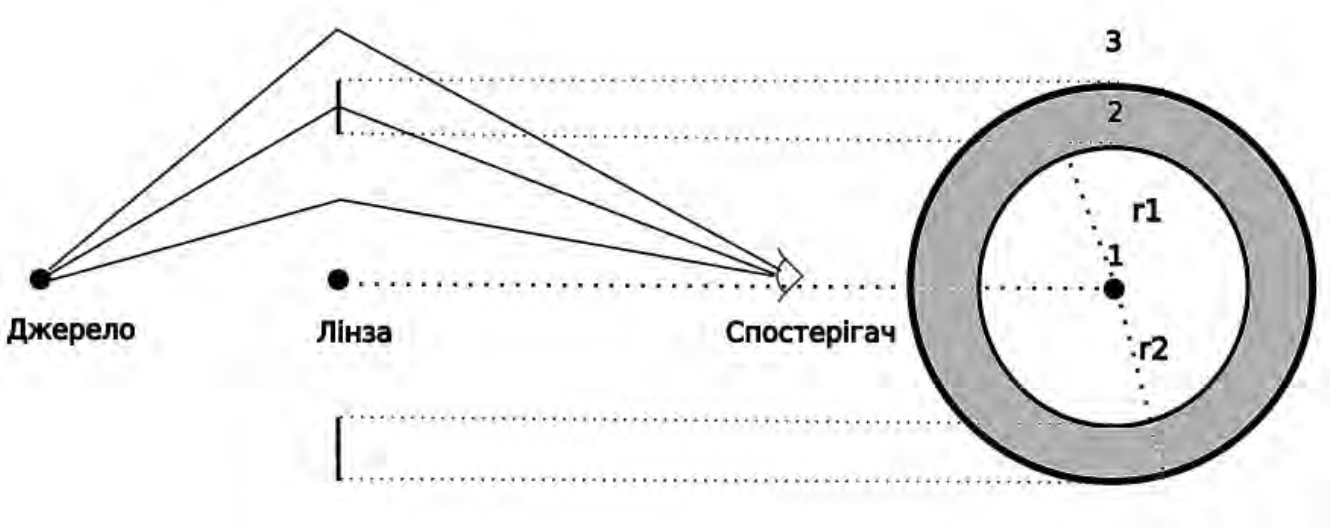


Рис. 18: Схема однорідного тора з центральною масою

Маса диска m_d та центральна маса m_0 обезрозмірені та нормованні на повну масу системи. Таким чином:

$$m_0 + m_d = 1. \tag{15}$$

Тоді рівняння лінзи має вигляд [5]

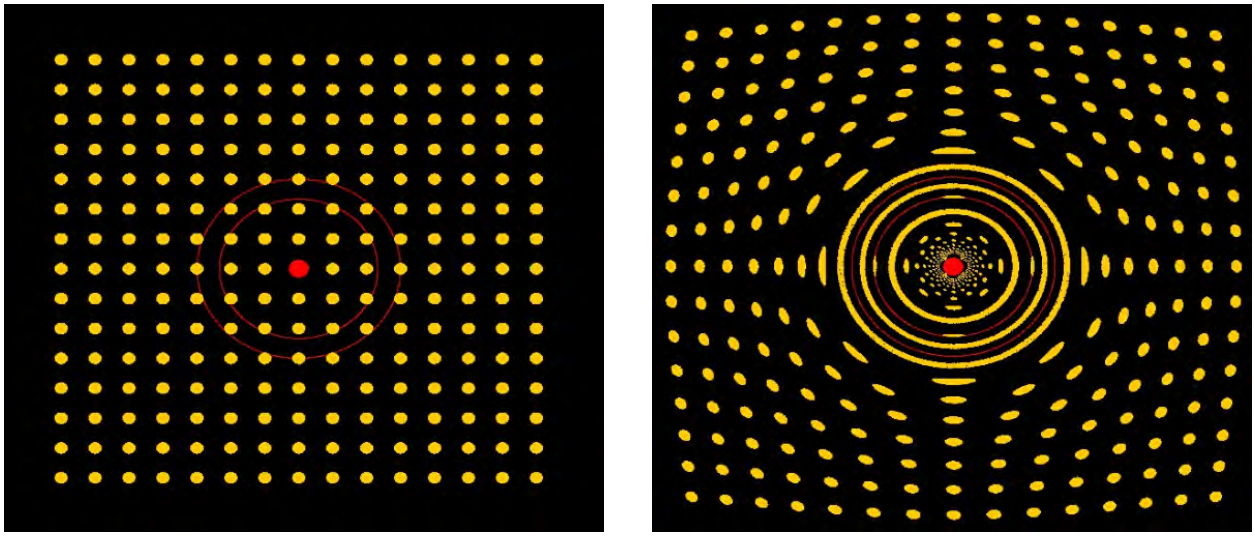
$$\vec{y} = \vec{x} \begin{cases} (1 - \dfrac{1}{x^2}) & x \geq r_2 \quad (1) \\ (1 - \dfrac{m_0}{x^2} - \kappa \dfrac{x^2 - r_1^2}{x^2}), & r_1 \leq x \leq r_2 \quad (2) \\ (1 - \dfrac{m_0}{x^2}) & x \leq r_1 \quad (3) \end{cases} \tag{16}$$

де $\kappa = m_d/(r_2^2 - r_1^2)$.

Рішення цієї системи відносно $\vec{x}$ має вигляд:

$$\vec{x} = \vec{y} \begin{cases} \frac{1}{2}\left(1 \pm \sqrt{y + \frac{4}{y}}\right) & x \geq r_2 \quad (1) \\ \frac{1 \pm \sqrt{1 + \frac{4(\kappa-1)(m_0 - \kappa r_1^2)}{y^2}}}{2(\kappa - 1)}, & r_1 \leq x \leq r_2 \quad (2) \\ \frac{1}{2}\left(1 \pm \sqrt{y + \frac{4m_0}{y}}\right) & x \leq r_1 \quad (3) \end{cases} \tag{17}$$

З системи рівняннь (17) бачимо, що одне джерело може мати до шести зображень. Використовуючи рівняння (17) за допомогою створеної програми можемо отримувати зображення гравітаційного лінзування на цій системі (Рис 19). В додатку А приведено більше зображень та графіків лінзування для цієї системи.



(а) (б)

Рис. 19: На рисунку (а) зображено квадратну решітку джерел світла 15 на 15. На рисунку (б) відповідні зображення. Червона точка позначає центральну масу, а червоні кола внутрішню та зовнішню межу диску. Параметри лінзи $m_0 = 0.3, r_1 = 0.7, r_2 = 0.9$

Також використовуючи систему рівняннь (16) можемо побудувати графік кута відхилення в залежності від кординати зображення – Рис. 20

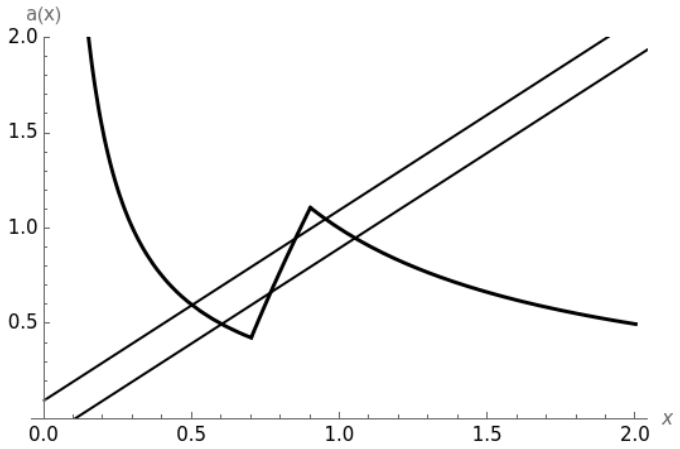


Рис. 20: Приклад залежністі кута відхилення від координати зображення для системи з параметрами $m_0 = 0.3, r_1 = 0.7, r_2 = 0.9$

Тут ширина області обмеженої чорними лініями рівна діаметру джерела світла, що в данному випадку рівний 0.1. Кількість перетенів графікому цієї області відовідає кількості кілець Ейнштейна, що утворюються, а протяжність перетину по x за товщину кілець.

Щоб знайти умови при яких зображення зливаються чи зникають розглянемо якобіан переходу.

$$\det A = \begin{cases} (1 - \dfrac{1}{x^4}) & x \geq r_2 \quad (1) \\ (1-\kappa)^2 - (\dfrac{\kappa r_2^2 - 1}{x^2})^2, & r_1 \leq x \leq r_2 \quad (2) \\ \left(1 - \dfrac{m_0^2}{x^4}\right) & x \leq r_1 \quad (3) \end{cases} \tag{18}$$

Розв'язками рівняння $\det A = 0$ є:

$$\begin{cases} x = \sqrt{m} & x \leq r_1 \quad (\text{A}) \\ x = r_1, & (\text{B}) \\ x = \sqrt{\dfrac{\kappa r_2^2 - 1}{1-\kappa}}, & r_1 \leq x \leq r_2 \quad (\text{C}) \\ x = r_2, & (\text{D}) \\ x = 1 & x \geq r_2 \quad (\text{E}) \end{cases} \tag{19}$$

І використовуючи рівняння (16) знайдемо відповідні каустики:

$$\begin{cases} y = 0 & \text{(A)} \\ y = r_1 - \dfrac{m_0}{r_1} & \text{(B)} \\ y = 0 & \text{(C)} \\ y = r_2 - \dfrac{1}{r_2} & \text{(D)} \\ y = 0 & \text{(E)} \end{cases} \tag{20}$$

Приклад залежності $\det A(x)$ приведено на Рис. 21.

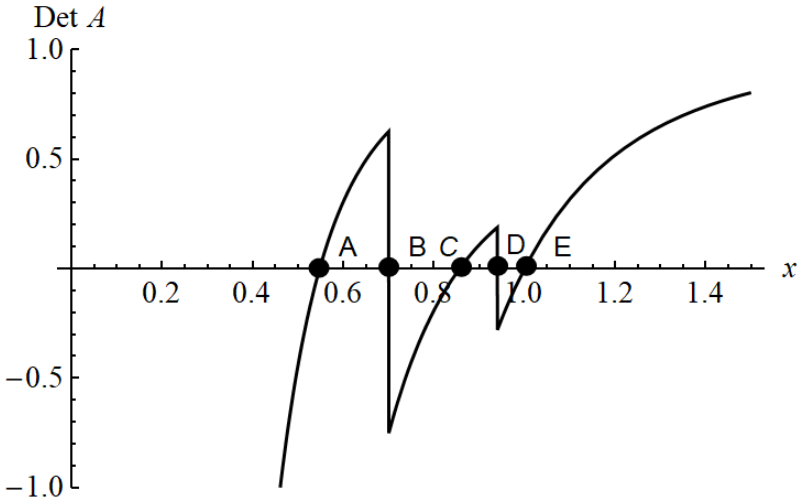
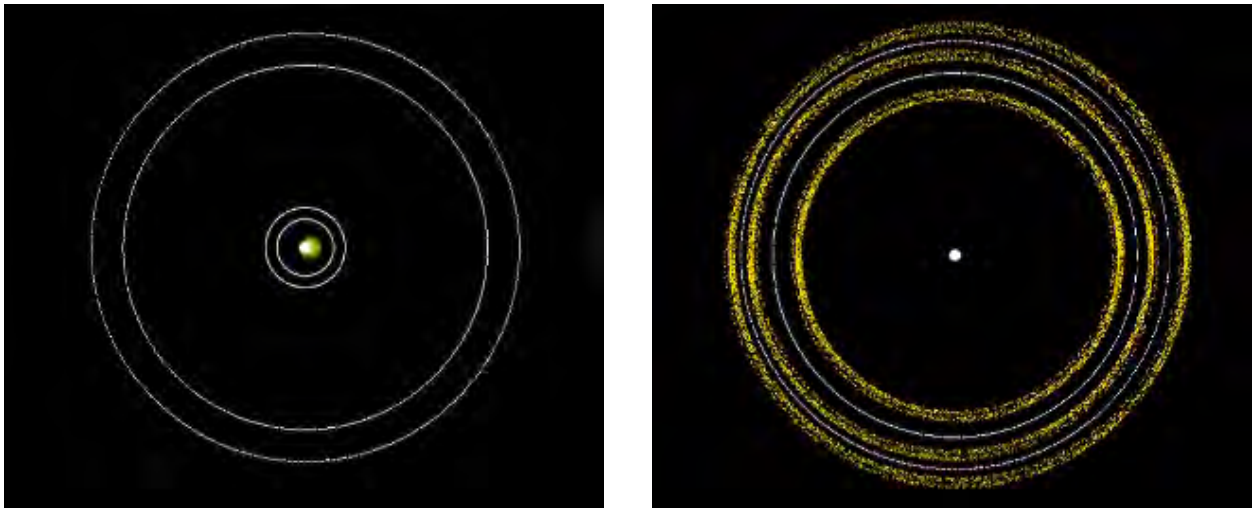


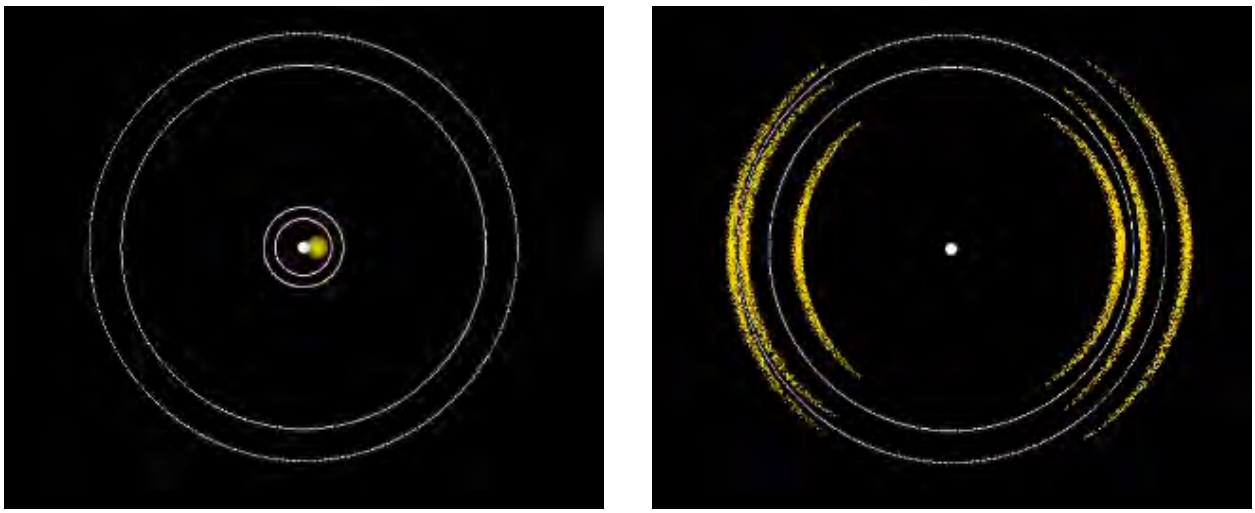
Рис. 21: Залежність детермінанта якобіана перетворення від координати зображення для $m_0 = 0.5, r_1 = 0.7, r_2 = 0.94$.

У загальному випадку в рівнянні (19) маємо три криві (A), (C), і (E), які у площині джерела світла переходять у точку 0, отже вони відповідають за утворення кілець Ейнштейна. Криві (B) і (D) у площині джерела утворюють каустики які відповідають за злиття зображень в областях 1 і 2 (каустика (B)) і в областях 2 і 3 (каустика (D)). Це проілюстровано на Рис. 22-27 для системи з параметрами $m_0 = 0.5, r_1 = 0.8, r_2 = 0.94$



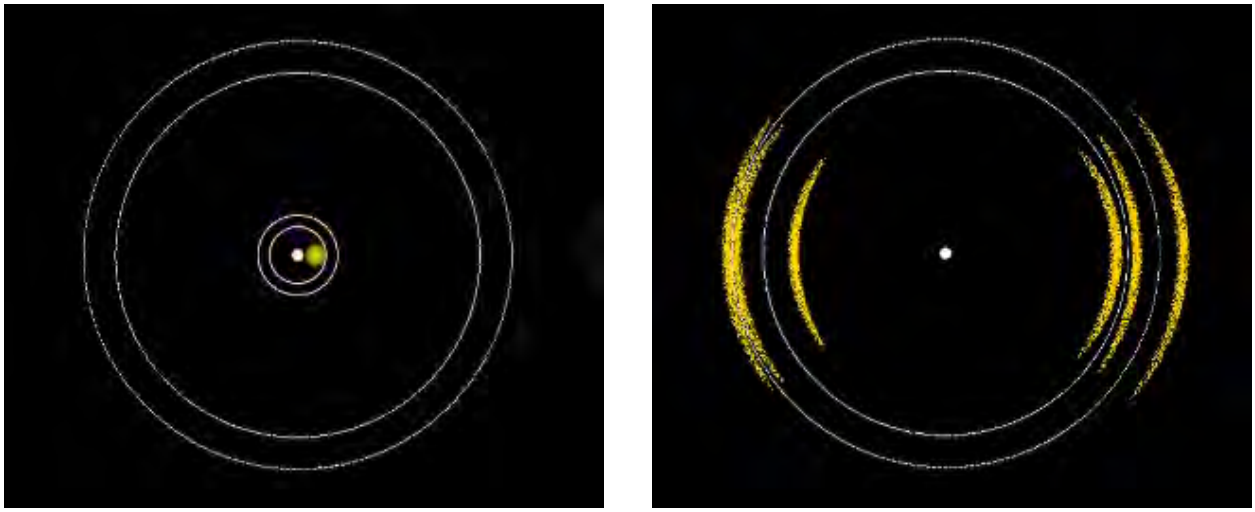
(а) (б)

Рис. 22: На рисунку (а) зображено джерело світла, що знаходиться у центрі координатю. Менші кола в середені є каустиками B і D. На рисунку (б) бачимо, що утворилось зображення 3 кілець Ейнштейна. На рисунку (а) і (б) великі кола позначають внутрішню та зовнішню межу диску, а точка в центрі, є центром координат де знаходиться центральна маса.



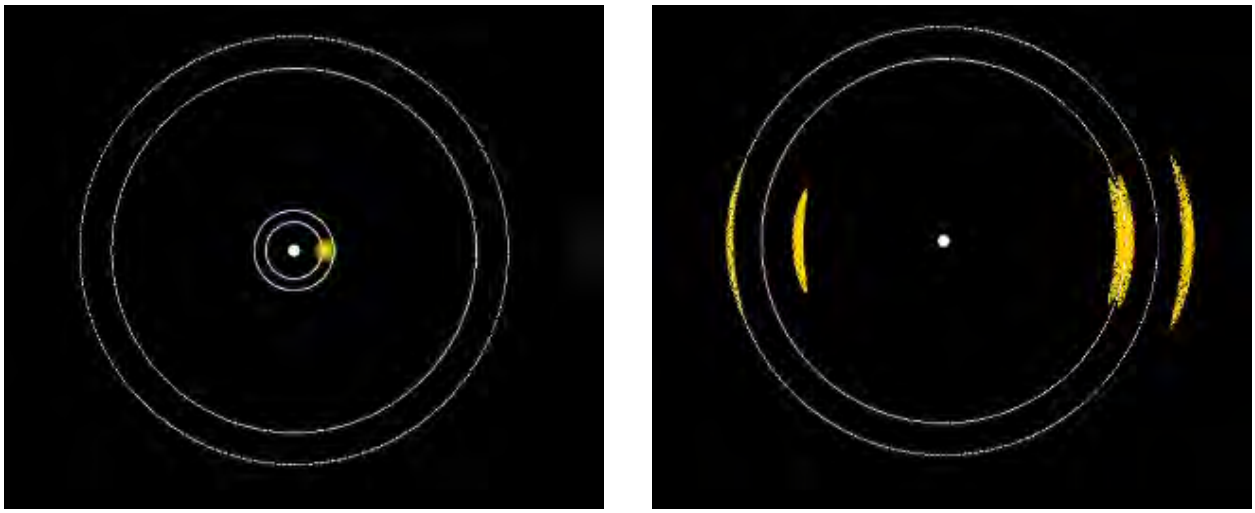
(а) (б)

Рис. 23: На рисунку (а) джерело світла вже не торкається центру координат. На рисунку (б) зображення трьох кілець Ейнштейна розпались на 6 дуг. Числом будемо позначати область де утворилось зображення. Зправа знаходяться зображення з позитивною парністю, а зліва з негативною.



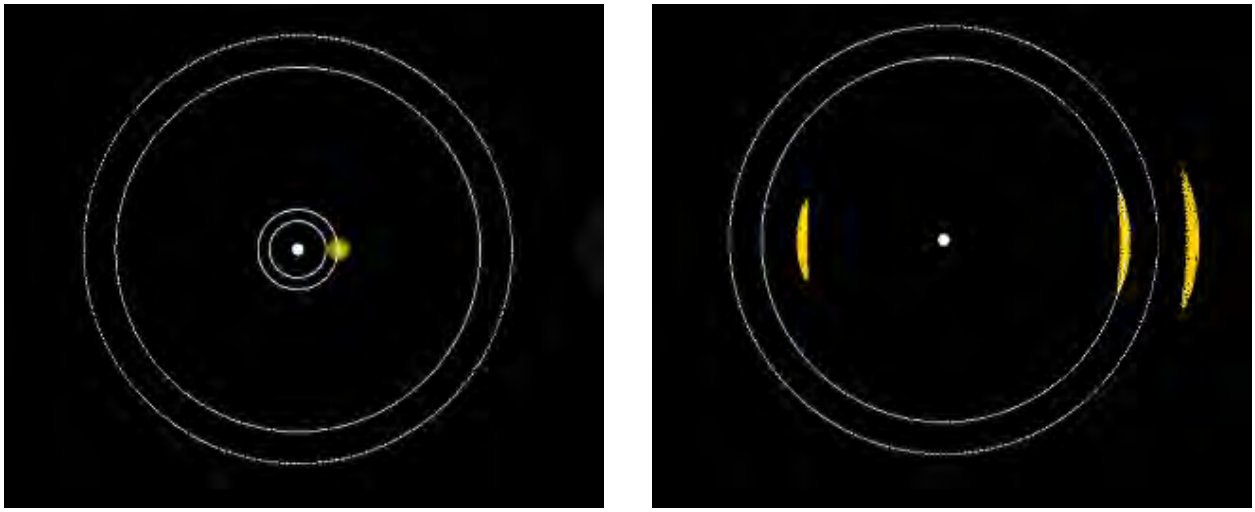
(а) (б)

Рис. 24: На рисунку (а) джерело світла починає перетинати першу каустику На рисунку (б) зображення 3- та 2- починають зливатися.



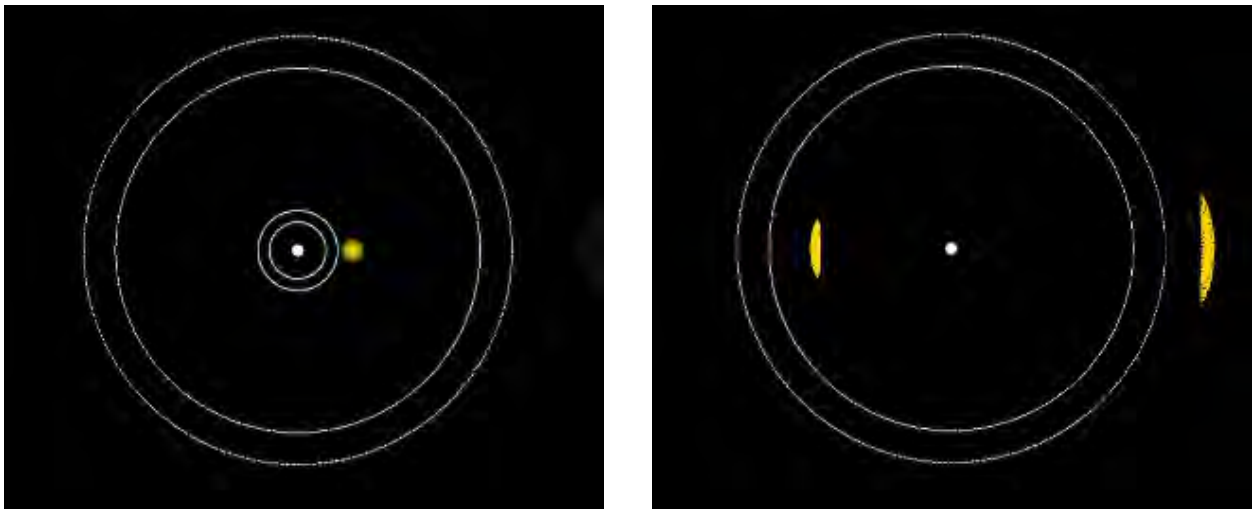
(а) (б)

Рис. 25: На рисунку (а) джерело світла перетинає обидві каустики. На рисунку (б) зображення 3- та 2- стають меншими, а зображення 1+ та 2+ починають зливатися.



(а) (б)

Рис. 26: На рисунку (а) джерело світла вже перетнуло першу каустику, та все ще перетинає другу каустику. На рисунку (б) Зображення 3- та 2- повністю зникають, а зображення 1+ та 2+ стають меншими.



(а) (б)

Рис. 27: На рисунку (а) джерело світла вже перетнуло другу каустику. На рисунку (б) Зображення 1+ та 2+ повністю зникають та залишаються тільки 1- та 3+.

На Рис. 28 бачимо графіки залежності координат зображень від координат джерела світла: числом показана область де утворюється зображення. Плюс(мінус) відповідає за зображення з позитивною(негативною) парністю. Насправді зображення з позитивною і негативною парністю з'являються з протилежних сторін лінзи, проте тут приведено лише модулі координат, для того, щоб легше побачити пари яких зображень утворюють кільця. Координати x при

$y = 0$, де дві дуги зливається і є радіусами кілець Ейнштейна. При віддаленні джерела світла від центра координат кожне кільце розпадається на дві дуги, що потім можуть злитися і зникнути при перетені джерелом каустик.

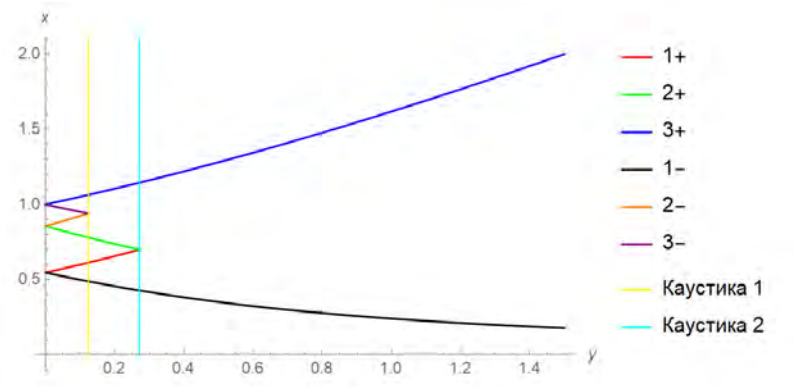


Рис. 28: Залежність модуля координат зображення від модуля координат джерела для $m_0 = 0.3, r_1 = 0.7, r_2 = 0.9$.

Використовуючи рівняння (11), (16) і (18) можемо отримати коефіцієнт підсилення в кожній області:

$$\mu_1^{\pm} = \frac{1}{4}\frac{\left(y \pm \sqrt{y^2 + 4m_0}\right)^2}{y\sqrt{y^2 + 4m_0}} \tag{21}$$

$$\mu_2^{\pm} = \frac{1}{4(\kappa-1)^2}\frac{\left(y \pm \sqrt{y^2 + 4(\kappa-1)(\kappa r_2^2 - 1)}\right)^2}{y\sqrt{y^2 + 4(\kappa-1)(\kappa r_2^2 - 1)}} \tag{22}$$

$$\mu_3^{\pm} = \frac{1}{4}\frac{(y \pm \sqrt{y^2+4})^2}{y\sqrt{y^2+4}}, \tag{23}$$

а повний коефіцієнт підсилення визначається як:

$$\mu = \mu_1^+ + \mu_1^- + \mu_2^+ + \mu_2^- + \mu_3^+ + \mu_3^- \tag{24}$$

Гравік залежності коефієнта підсилення від координати джерела приведено на Рис. 29

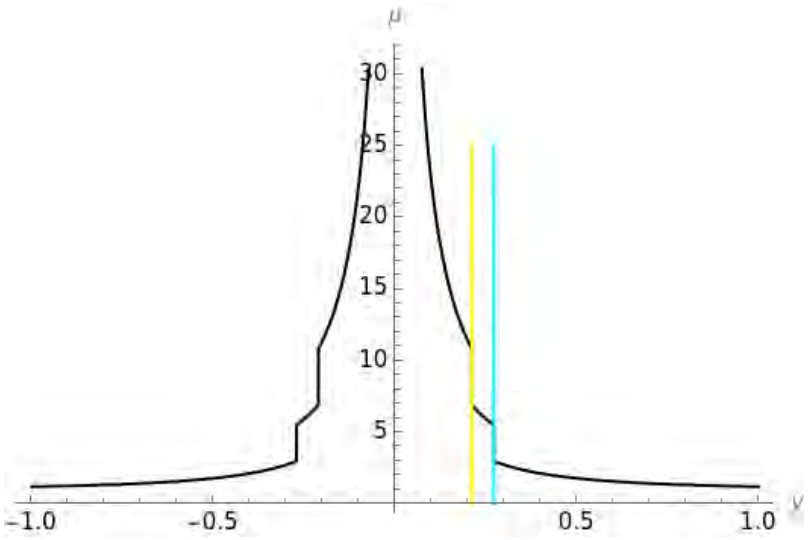


Рис. 29: Графік залежності коефіцієнта підсилення від координати зображення для системи з параметрами $m_0 = 0.3, r_1 = 0.7, r_2 = 0.9$

Бачимо, що проходячи через каустики коефіцієнт підсилення стрибком змінюється. Це пов'язано з тим, що два зображення зникають.

Графіки кута відхилення, детермінанта та коефіцієнта підсилення несуть інформацію про кількість зображень та дають можливість побачити як різні ефекти гравітаційного лінзування пов'язані оде з одним.

3.2 Розрахунок рівняння тора в термінах R_{tor} і r_0

Розглянемо розподіл поверхневої густини у диску виражений формулою (25), що імітує однорідний тор (дивись Рис. (30))

$$\rho(\xi) = \rho_d \sqrt{1 - \left(\frac{\xi}{r_0} - \frac{R_{tor}}{r_0}\right)^2}. \tag{25}$$

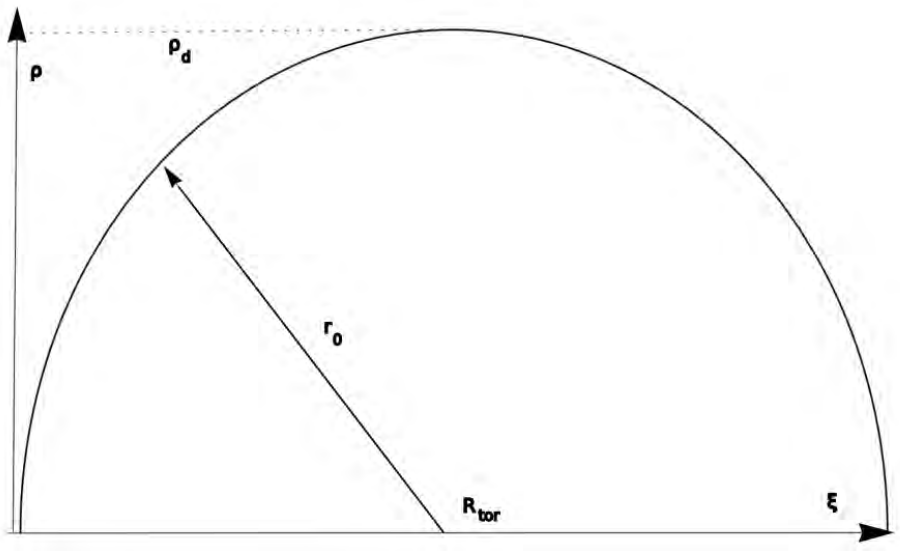


Рис. 30: Розподіл густини, що імітує однорідний тор

Розрахуємо масу тора:

$$M_d = 2\pi \int_{R_{tor}-r_0}^{R_{tor}+r_0} \xi\rho(\xi)d\xi = 2\pi\rho_d I_1\,,$$

де $I_1 = \int_{R_{tor}-r_0}^{R_{tor}+r_0} \xi\sqrt{1-\left(\frac{\xi}{r_0}-\frac{R_{tor}}{r_0}\right)^2}d\xi$.
Введемо заміну

$$\sin\zeta = \frac{\xi}{r_0} - \frac{R_{tor}}{r_0} \to \xi = r_0\sin\zeta + R_{tor} \to \cos\zeta d\zeta = \frac{1}{r_0}d\xi$$

Межі інтегрування зміняться наступним чином:
При $\xi = R_{tor} - r_0$ маємо $\sin\zeta = \frac{R_{tor}-r_0-R_{tor}}{r0} = -1 \to \zeta = -\frac{\pi}{2}$
При $\xi = R_{tor} + r_0$ маємо $\sin\zeta = \frac{R_{tor}+r_0-R_{tor}}{r0} = 1 \to \zeta = \frac{\pi}{2}$ Інтеграл I_1 перетворюється у

$$\int_{-\frac{\pi}{2}}^{\frac{\pi}{2}} \cos^2\zeta(r_0\sin\zeta + R_{tor})r_0 d\zeta = r_0^2\int_{-\frac{\pi}{2}}^{\frac{\pi}{2}} \cos^2\zeta\sin\zeta d\zeta + r_0 R_{tor}\int_{-\frac{\pi}{2}}^{\frac{\pi}{2}} \cos^2\zeta d\zeta$$

Розрахуємо кожен інтеграл окремо. Використовуючи формулу пониження степе-

ня $\cos^2\zeta = \frac{1+\cos 2\zeta}{2}$

$$r_0 R_{tor} \int_{-\frac{\pi}{2}}^{\frac{\pi}{2}} \frac{1+\cos 2\zeta}{2} d\zeta = \frac{r_0 R_{tor}\zeta}{2} + \frac{r_0 R_{tor}}{4}\sin 2\zeta \Big|_{-\frac{\pi}{2}}^{\frac{\pi}{2}} = \frac{r_0 R_{tor}}{2}\pi$$

$$r_0 \int_{-\frac{\pi}{2}}^{\frac{\pi}{2}} \cos^2\zeta \sin\zeta d\zeta = -r_0 \int_{-\frac{\pi}{2}}^{\frac{\pi}{2}} \cos^2\zeta d\cos\zeta = -\frac{r_0^2}{3}\cos^3\zeta \Big|_{-\frac{\pi}{2}}^{\frac{\pi}{2}} = 0$$

Таким чином

$$I_1 = \frac{r_0 R_{tor}}{2}\pi \tag{26}$$

$$M_d = 2\pi\rho_d \frac{r_0 R_{tor}}{2}\pi = R_{tor} r_0 \rho_d \pi^2 \to \rho_d = \frac{M_d}{R_{tor} r_0 \pi^2} \tag{27}$$

Тепер розрахуємо кут відхилення тора.

$$\alpha_d(x) = \frac{2}{x}\int \kappa(x')x'dx'$$

де $\kappa(x') = \frac{\rho(\xi_0 x')}{\rho_{cr}}$ і $\rho_{cr} = \frac{M_0+M_d}{\pi\xi_0}$ Тоді

$$\kappa(x') = \frac{\pi\xi_0^2}{M_0+M_d}\frac{M_d}{r_0 R_{tor}\pi^2}\sqrt{1-\left(\frac{\xi_0 x'}{r_0} - \frac{R_{tor}}{r_0}\right)^2}$$

R_{tor} та r_0 нормуємо на радіус кільця Ейнштейна ξ_0 і нормуємо маси як у формулі (15). Таким чином безрозмірна поверхнева густина має вид:

$$\kappa_d = \frac{m_d}{\pi r_0 R_{tor}}$$

І отримаємо

$$\alpha_d(x) = \frac{2\kappa_d}{x} I_2(x)$$

$$I_2(x) = \int_{R_{tor}-r_0}^{x} x' \sqrt{1 - \left(\frac{x'}{r_0} - \frac{R_{tor}}{r_0}\right)^2}\, dx'$$

Інтеграл I_1 і I_2 одного виду, тож можимо записати

$$I_2(x) = \frac{r_0 R_{tor}\zeta}{2} + \frac{r_0 R_{tor}}{4}\sin 2\zeta - \frac{r_0^2}{3}\cos^3\zeta\Big|_{-\frac{\zeta}{2}}^{\frac{\zeta}{2}} =$$

$$= \frac{r_0 R_{tor}\zeta}{2} + \frac{r_0 R_{tor}}{4}\sin 2\zeta - \frac{r_0^2}{3}\cos^3\zeta + \frac{\pi r_0 R_{tor}}{4} \quad (28)$$

$$I_2(x) = \frac{r_0 R_{tor}}{4}(2\zeta + \sin 2\zeta + \pi) - \frac{r_0^2}{3}\cos^3\zeta \quad (29)$$

Де

$$\zeta = \arcsin\left(\frac{x}{r_0} - \frac{R_{tor}}{r_0}\right) \quad (30)$$

Підставимо зовнішню межу тора

$$I_2(R_{tor}+r_0) = \frac{r_0 R_{tor}}{4}(2\arcsin 1 + \sin(2\arcsin 1) + \pi) - \frac{r_0^2}{3}\cos^3\arcsin 1 = \frac{\pi r_0 R_{tor}}{2} = I_1$$

Тоді нарешті отримаємо рівняння лінзу для однорідного тора.

$$\vec{y} = \vec{x}\begin{cases} (1 - \frac{1}{x^2}) & x \geq R_{tor} + r_0 \quad (1) \\ (1 - \frac{m_0}{x^2} - \frac{2\kappa_d}{x^2} I_2(x), & R_{tor} - r_0 \leq x \leq R_{tor} + r_0 \quad (2) \\ (1 - \frac{m_0}{x^2}) & x \leq R_{tor} - r_0 \quad (3) \end{cases} \quad (31)$$

Отже рівняння лінзи у 1 та 3 області такі ж як і у випадку однорідного диску, зміни з'являться тільки у рівнянні для другої області. Отримане рівняння виражене у елементарних функціях, проте тут у рівняння входить arcsin і обернені рівняння отримати не вдається, тож для отримання зображень необхідно використовувати другий алгоритм описаний у розділі 2.1. Використовуючи ці

рівняння можна побудувати графік кута відхилення в залежності від координати зображення приведений на Рис. 31.

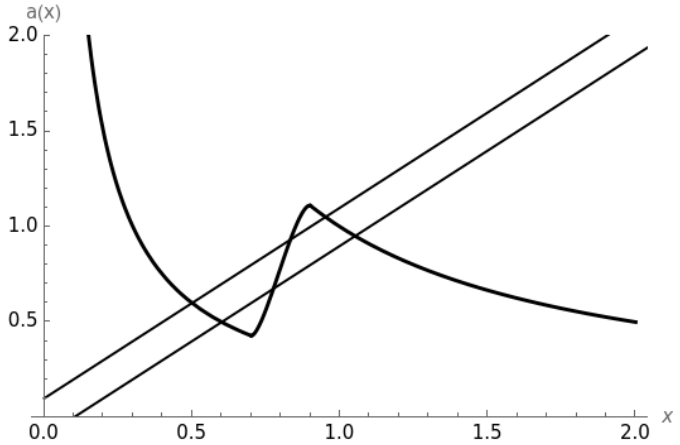
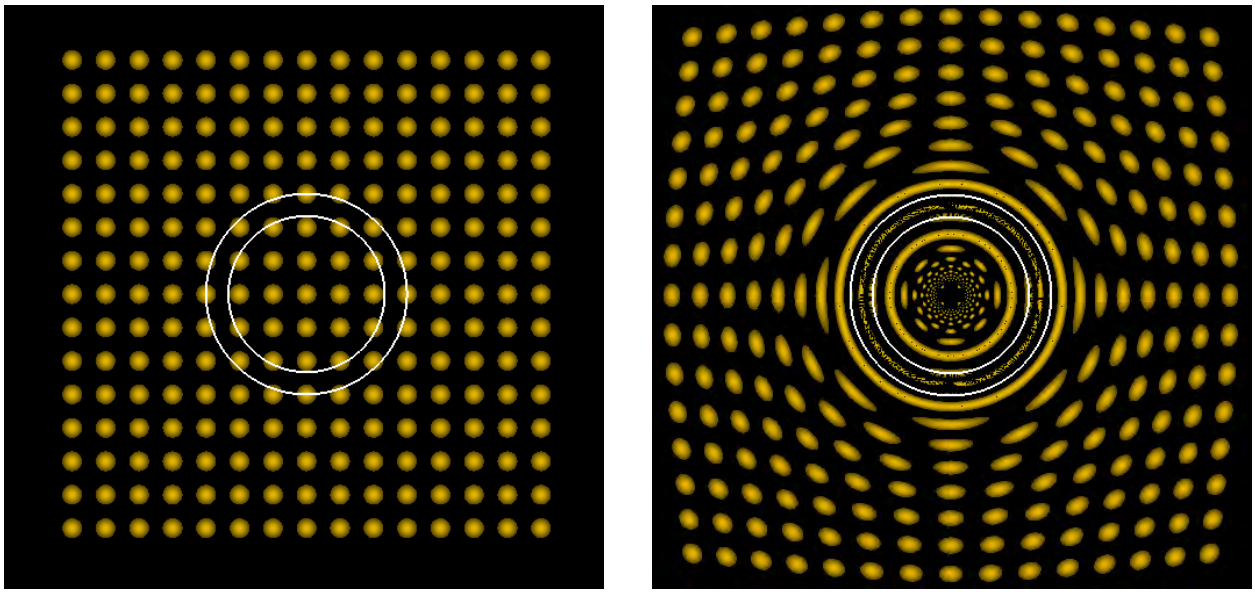


Рис. 31: Залежність кута відхилення від координати зображення для параметрів $m_0 = 0.3, R_{tor} = 0.8, r_0 = 0.1$

Гравік кута відхилення дуже подібний до того, що був отриманий для однорідного диску, проте на межах області мають більш плавний перегин. Також використовуючи рівння лінзи користуючи власною програмою було отримано зображення приведені на Рис. 32. В додатку А приведено більше зображень та графіків лінзування для цієї системи.



(а) (б)

Рис. 32: На рисунку (а) зображено квадратну решітку джерел світла 15 на 15. На рисунку (б) відповідні зображення. Білі кола внутрішню та зовнішню межу диску. Параметри лінзи $m_0 = 0.3, R_{tor} = 0.8, r_0 = 0.1$

Отриманне зображенняя також подібне до зображення утвореного при лінзуванні на однорідному диску.

Для 1 та 3 облаті залежність $\det A$ така сама як і для однорідного диску, проте у другій області вона визначається рівнянням (32)

$$\det A = 1 - \frac{m_0}{x} - \frac{2\kappa_d^2(r_0^2 - (R_{tor} - x)^2)(2r_0^2 + R_{tor}^2 + R_{tor}x - 2x^2)}{3r_0^2} + \frac{\kappa_d\sqrt{1 - \frac{(R_{tor}-x)^2}{r_0^2}}(2r_0^2 + R_{tor}^2 + 6m_0x + R_{tor}x - 8x^2)}{3x} + \frac{\kappa_d r_0 R_{tor}(-1 + 2\kappa_d x\sqrt{1 - \frac{(R_{tor}-x)^2}{r_0^2}})\arccos\frac{R_{tor}-x}{r_0}}{x} \quad (32)$$

Для такого громізького рівняння невдається знайти каустики. Графік залежності $\det A$ від координат джерела приведено на Рис. 33

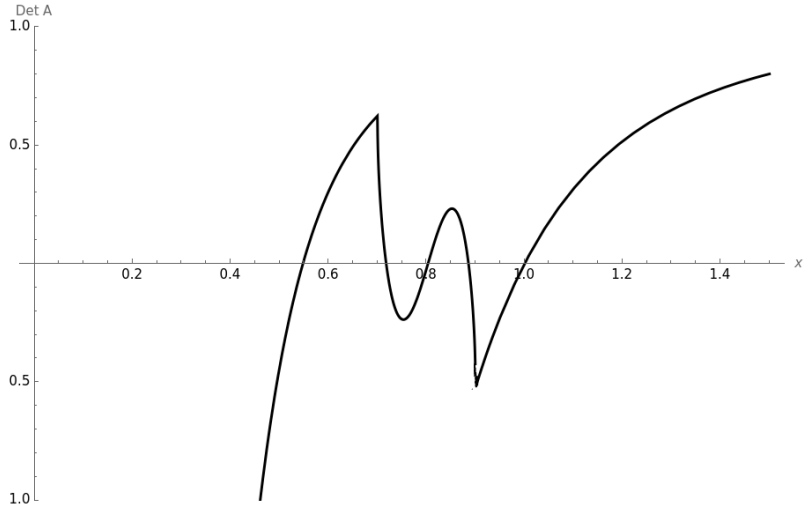


Рис. 33: Залежність детермінанта якобіана перетворення від координати зображення для $m_0 = 0.5, R_{tor} = 0.8, r_0 = 0.1$.

Цей графік також подібний до результатів моделювання для однорідного тиску, але виглядає згладженим.

На Рис. 34 преведено графік залежності координат зображення від координат джерела.

Також можемо помітити, що графік дуже подібний до випадку однорідного диску. Загалом для випадку тонкого тора, ефекти лінзування добре збігаються і

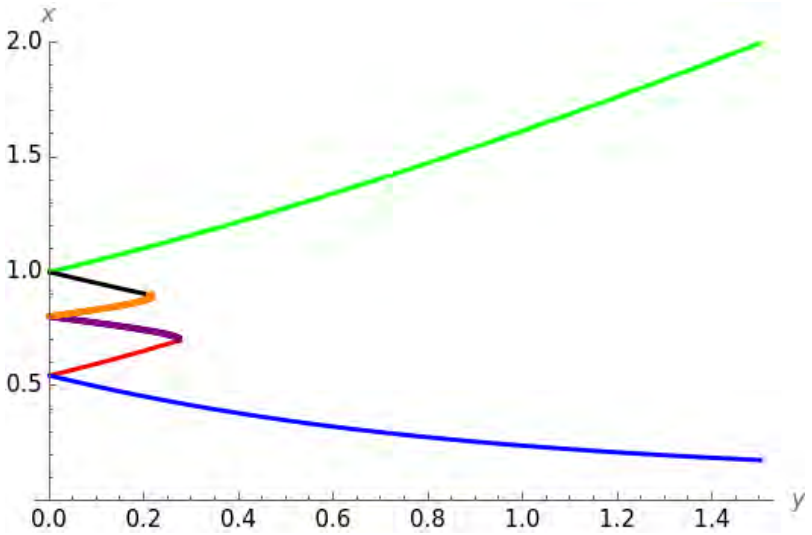
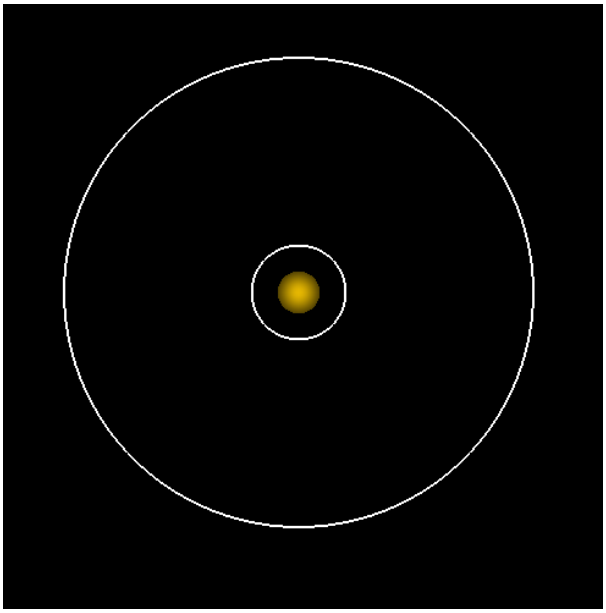


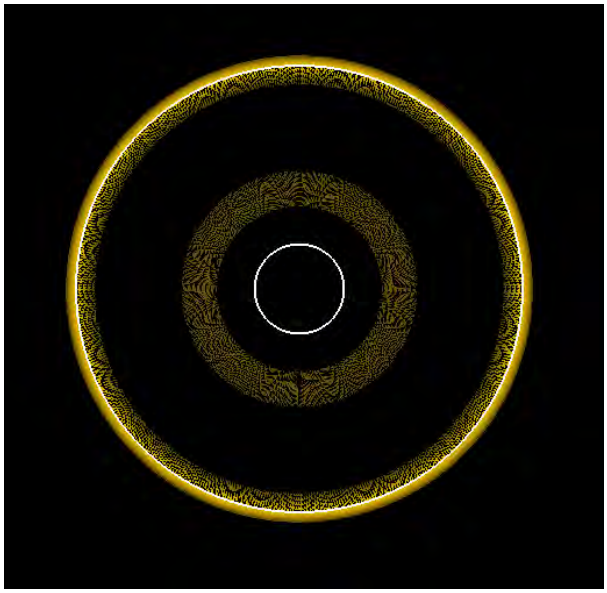
Рис. 34: Залежність координати зображення від координати джерела для $m_0 = 0.5, R_{tor} = 0.8, r_0 = 0.1$.

ця складна ситема може бути апроксимована однорідним диском. Проте у випадку товстого тора з'являється новий ефект - утворення двох кілець Ейнштейна в 2 області, що пов'язано з тим, що тепер критичні лінії проходять не по границям тора. Це продимонстровано на Рис. 35

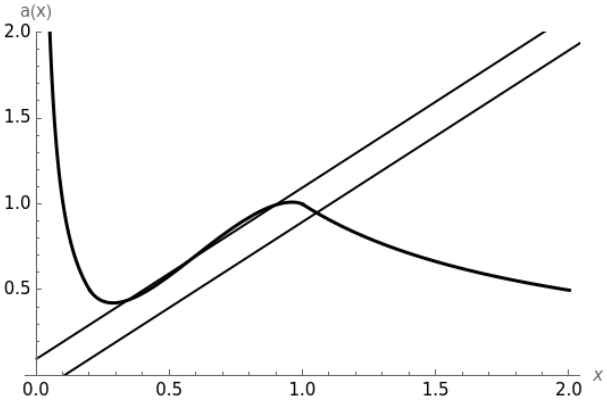
В майбутньому буде більш детально розібрано данну систему. Головним чином дослідження критичних кривих та умови утворення кілець Ейнштейна. Схоже, що додаткові зображення з'являються через зеркальний розподіл матерії. Також буде проведено роботу над знаходженням коефіцієнту підсилення, що ускладнюється громізкістю рівняння детермінанту у 2 області.



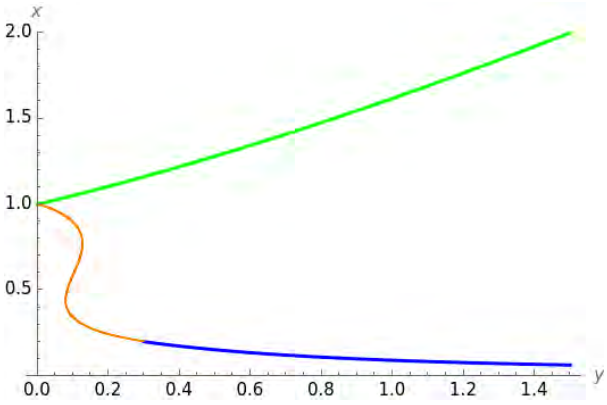
(а)



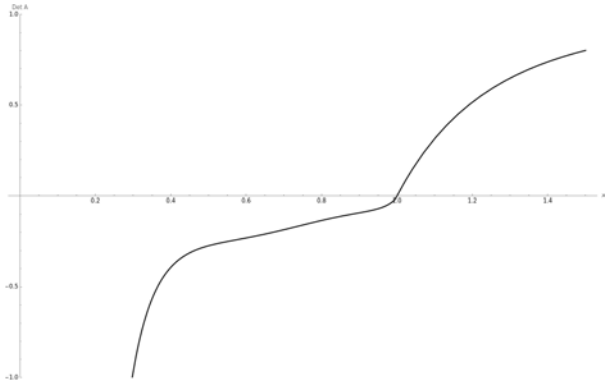
(б)



(в)



(г)



(д)

Рис. 35: Утворення двох кілець Ейнштейна в 2 області при параметрах системи $m_0 = 0.1, R_{tor} = 0.6, r_0 = 0.4$.

Висновки

В результаті виконаної роботи було досягнуто наступних результатів:

1. Розроблено власне програмне забезпечення для розрахунку та візуалізації ефектів гравітаційного лінзування.
2. Знайдено каустики та критичні лінії для системи однорідного диску з центральною масою.
3. Знайдено нове рівняння лінзи для системи, що імітує однорідний тор з центарльною масою.
4. Для випадку тонкого тора ефекти лінзування добре збігаються з тими, що ми спостерігаємо на системі однорідного диску.
5. Для товстого тора можливе утворення одразу двох кілець Ейнштейна в 2 області, що пов'язано з симетрією у розподілі речовини в торі.

В майбутньому робота над цією задачою буде продовжена. Наразі поставлено такі задачі.

1. Знайти коефіцієнт підсилення, критичні лінії та каустики для систем однорідний тор з центральною масою.
2. Розглянути нову системи, що імітує Гаусовий розподіл речовини у торі з центральною масою.
3. Для цієї системи отримати коефіцієнт підсилення, критичні лінії та каустики, а також залежність $\det A(x)$
4. Покращити власне програмне забезпечення для візуалізації ефектів гравітаційного лінзування.
5. Знайди залежність часових варіацій від координат зображення для описаних вище систем.

Додаток А: Зображення та графіки гравітаційного лінзування для різних систем

В цьому розділі приведені графіки гравітаційного лінзування на системі однорідного тора з центральною масою, та однорідного диску з центральною масою.

На Рис. 36-42 приведено зображення та графіки для системи однорідного диску з центральною масою. Тут (а) - джерело світла, білими колами показані каустики та розміри диску, біла точка відповідає центральній масі, (б) - відповідні зображення, білими колами показані розміри диску, біла точка відповідає центральній масі, (в) - залежність координат зображення від координат джерела, (г) - залежність кута відхилення від координати джерела, (д) - залежність детермінанта якобіана перетворення від координати зображення, (е) - залежність коефіцієнта підсилення від координати зображення.

На Рис. 43-49 приведено зображення та графіки для системи однорідного тору з центральною масою. Тут (а) утворенні в результаті лінзування зображення, білими колами показані розміри диску, біла точка відповідає центральній масі, (б) - залежність координат зображення від координат джерела, (в) - залежність кута відхилення від координати джерела, (г) - залежність детермінанта якобіана перетворення від координати зображення.

Для систем однорідний диск з центральною масою, та однорідний тор з центральною масою приведено графіки для систем, чиї параметри збігаються. Тобто на Рис. 36 приведено графіки для системи з параметрами $m_0 = 0.5, r_1 = 0.7, r_2 = 0.9$, а на Рис. 43 з параметрами $m_0 = 0.5, R_{tor} = 0.8, r_0 = 0.1$. Центральні маси рівні, маса дисків рівні, та їх геометричні розміри рівні. Це справедливо для кожної пари зображень. Таким чином можна детальніше відстежити, наскільке добре систему однорідного тора з центральною масою можна апроксимувати системою одрідний диск з центральною масою.

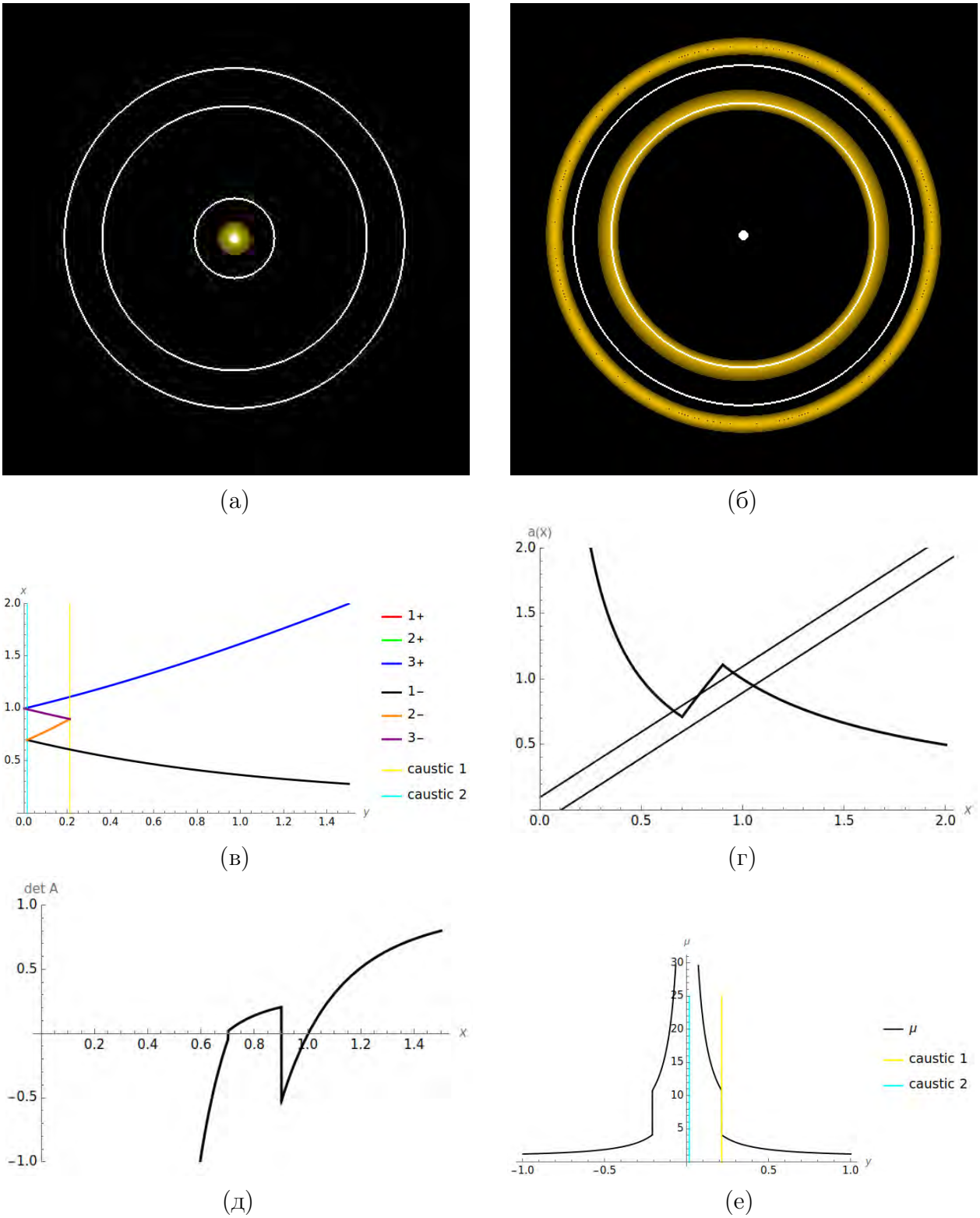


Рис. 36: Ефекти гравітаційного лінзування для системи з параметрами $m_0 = 0.5, r_1 = 0.7, r_2 = 0.9$

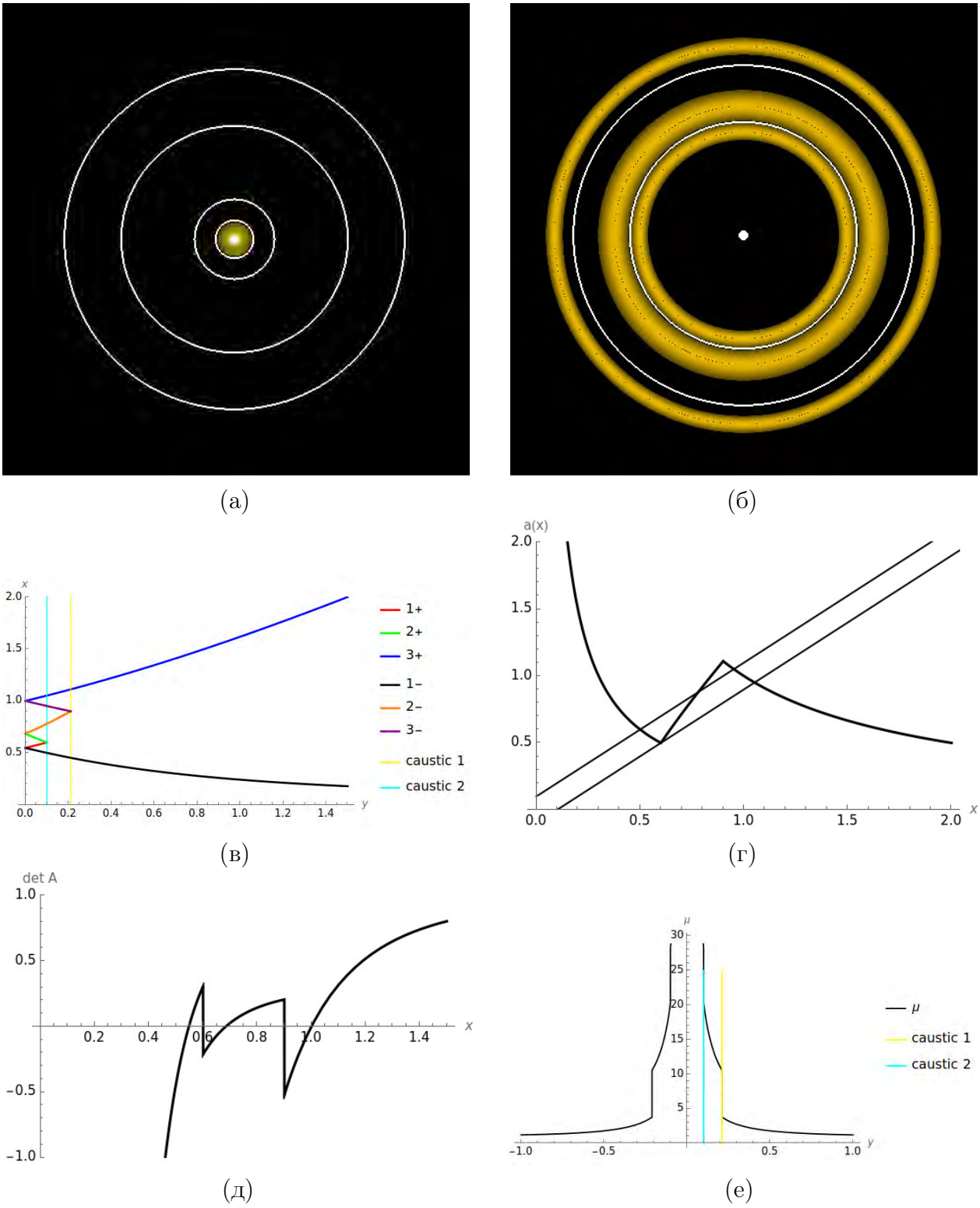


Рис. 37: Ефекти гравітаційного лінзування для системи з параметрами $m_0 = 0.3, r_1 = 0.6, r_2 = 0.9$

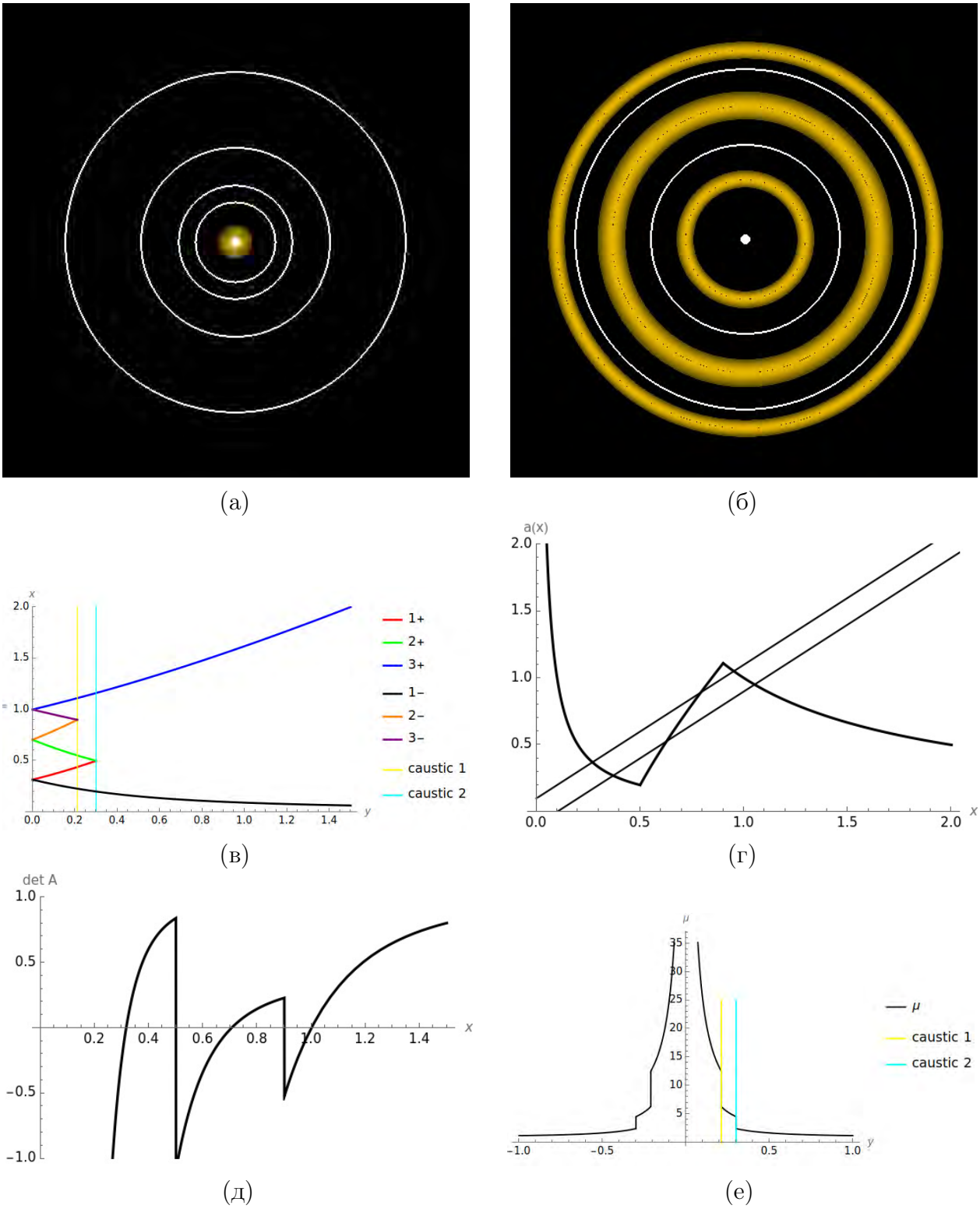
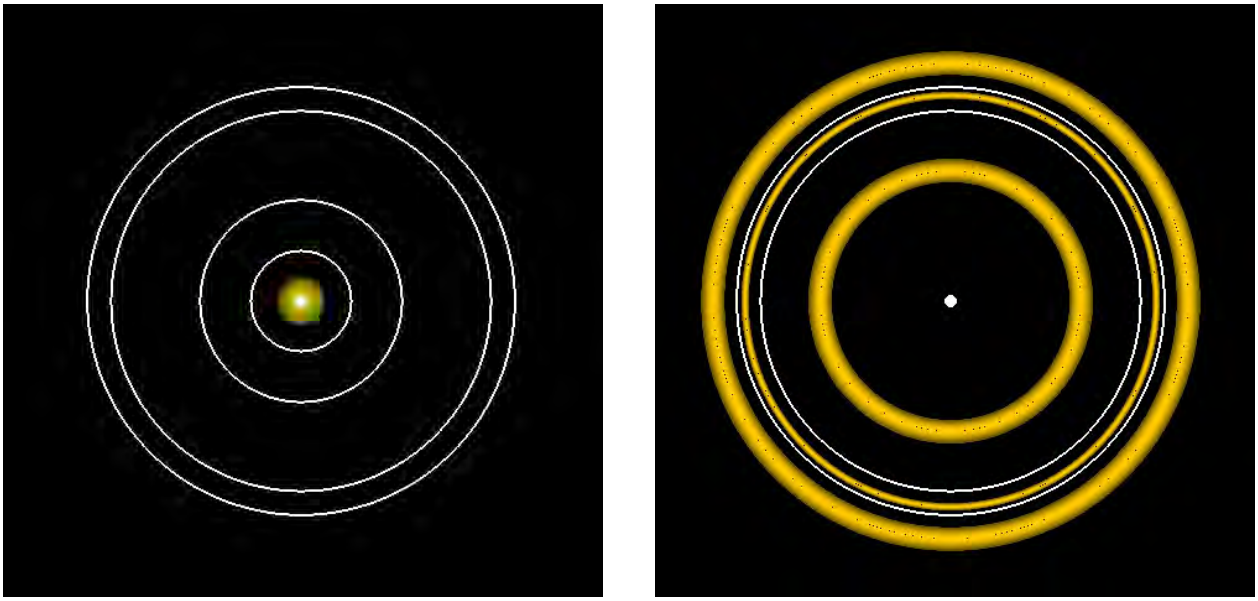
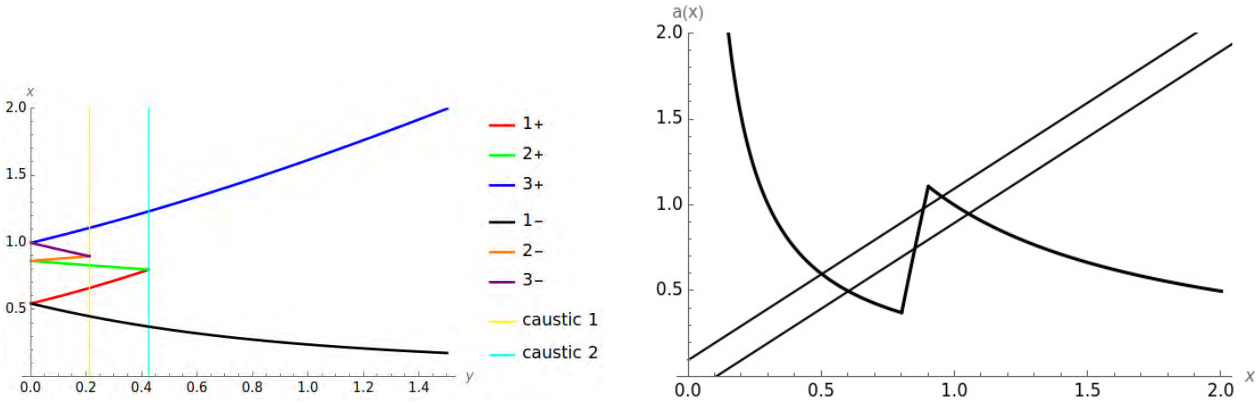


Рис. 38: Ефекти гравітаційного лінзування для системи з параметрами $m_0 = 0.1, r_1 = 0.5, r_2 = 0.9$

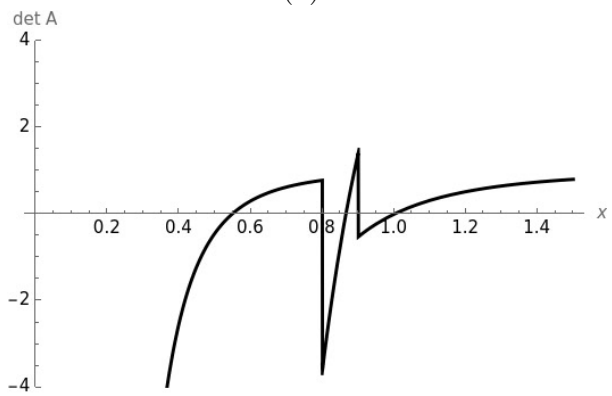


(а) (б)

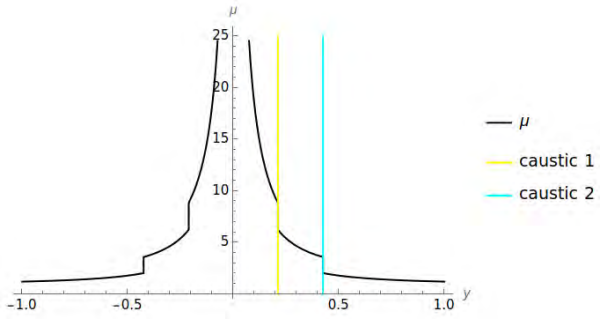


(в)

(г)

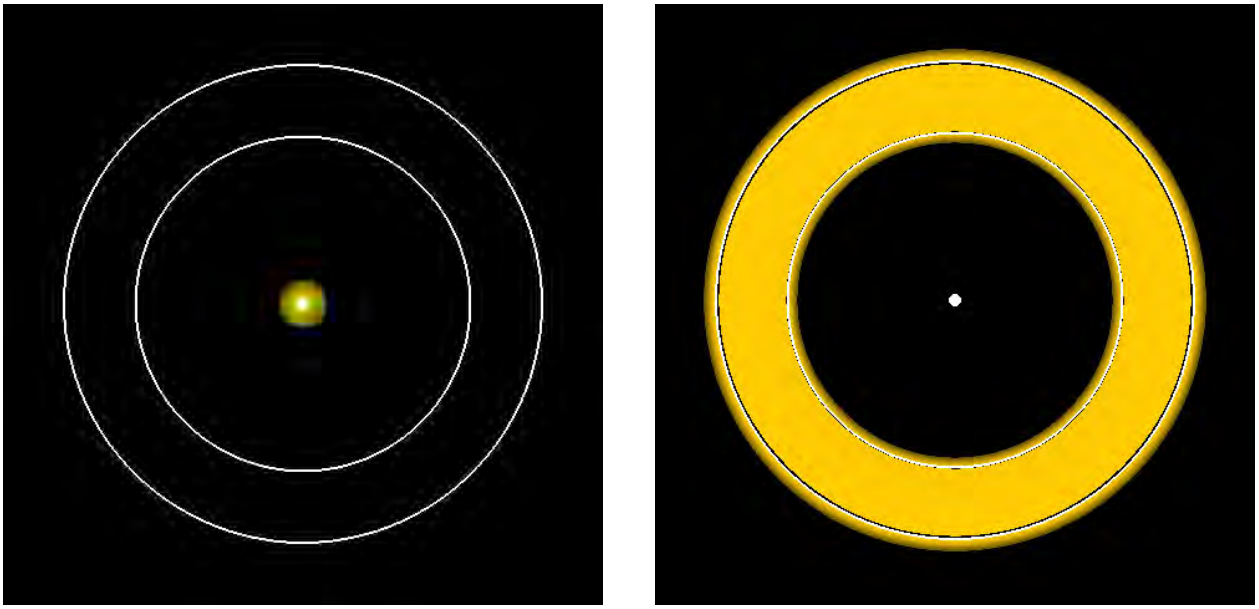


(д)

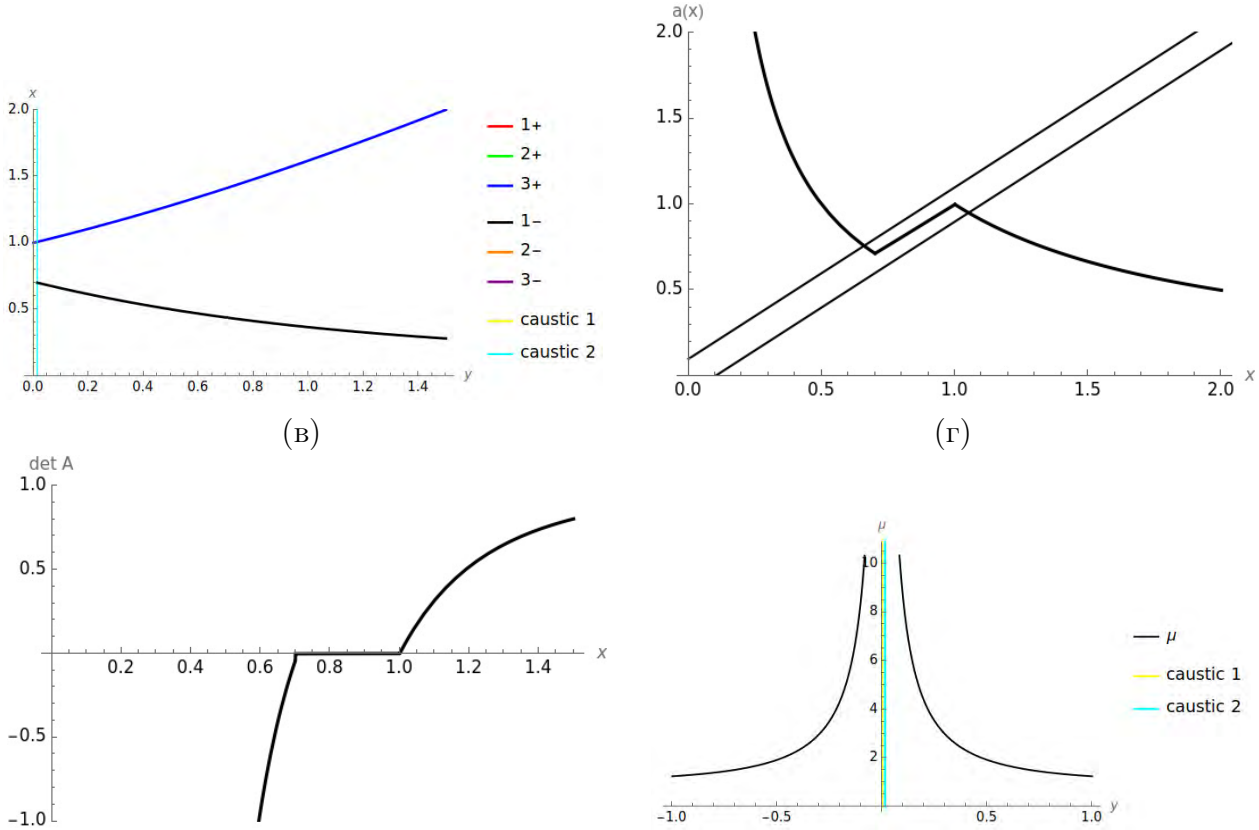


(е)

Рис. 39: Ефекти гравітаційного лінзування для системи з параметрами $m_0 = 0.3, r_1 = 0.8, r_2 = 0.9$



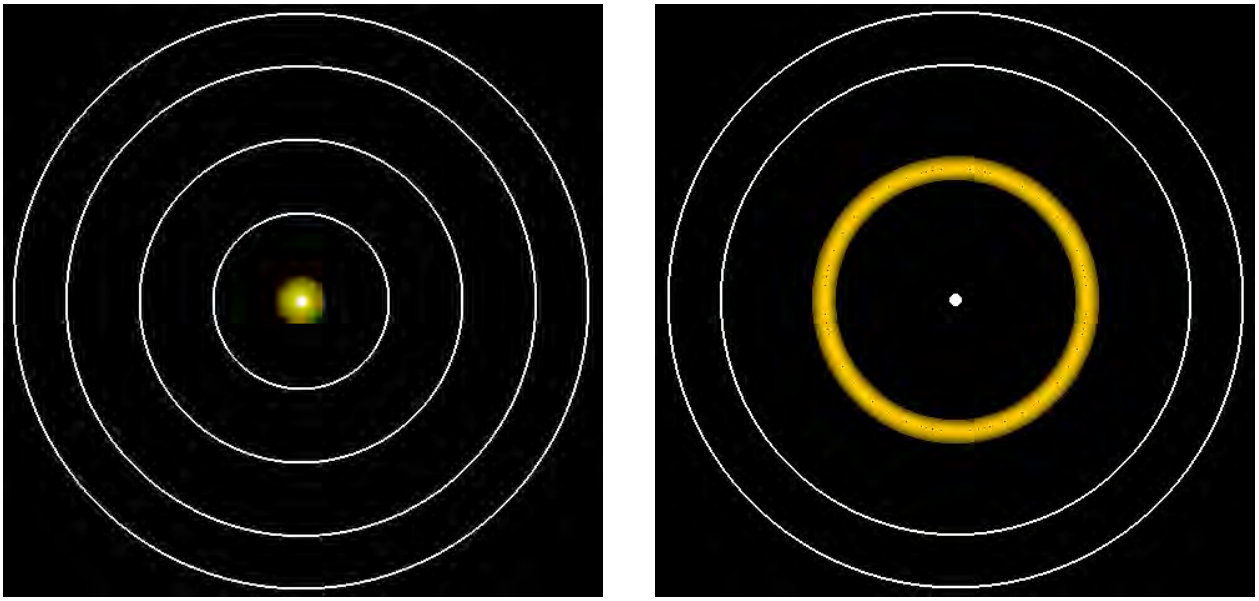
(а) (б)



(в) (г)

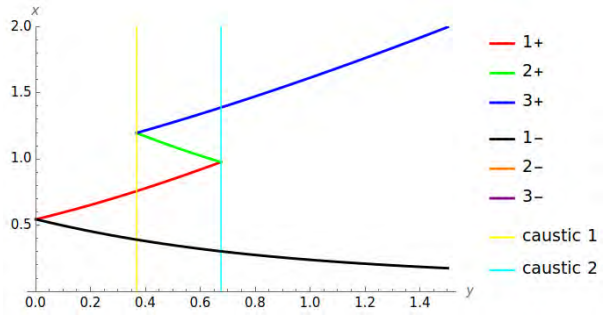
(д) (е)

Рис. 40: Ефекти гравітаційного лінзування для системи з параметрами $m_0 = 0.5, r_1 = 0.7, r_2 = 1$

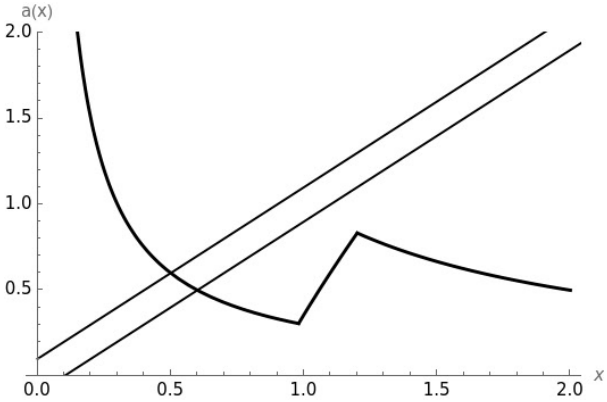


(а)

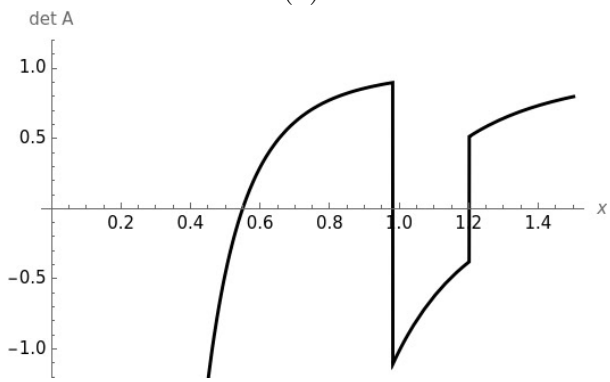
(б)



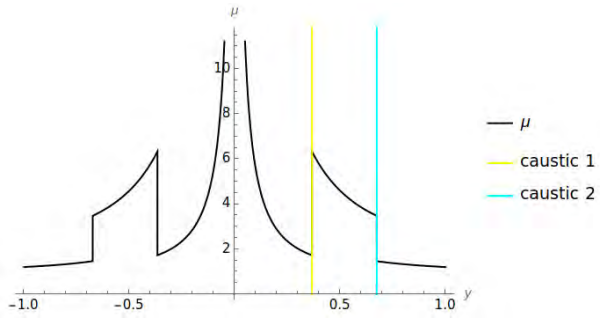
(в)



(г)

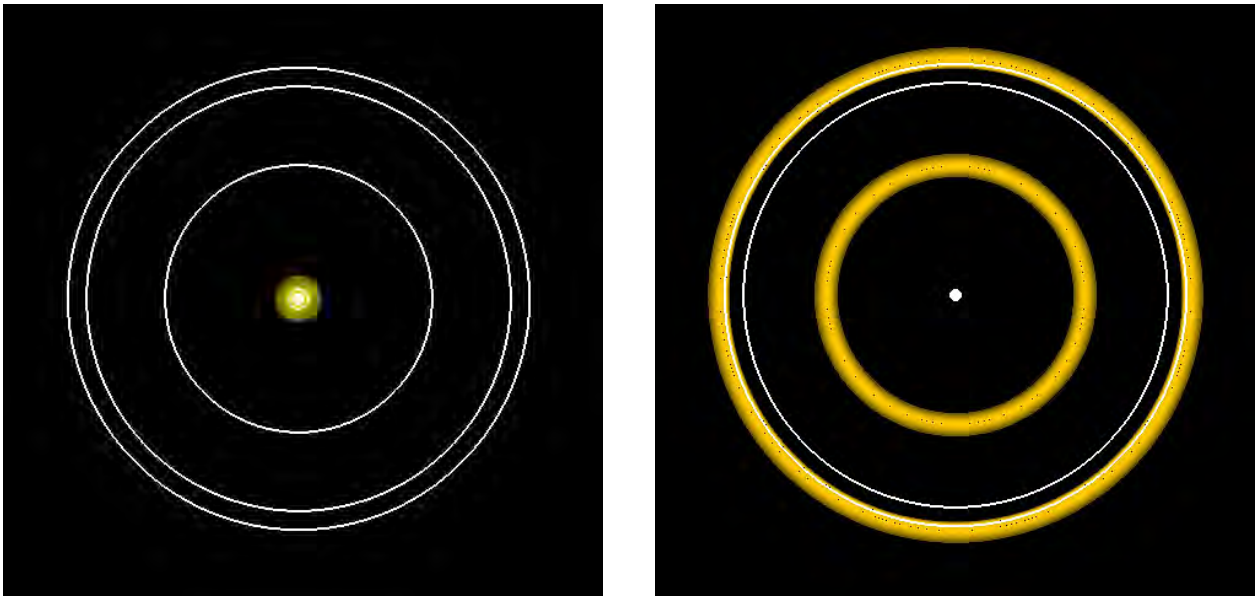


(д)

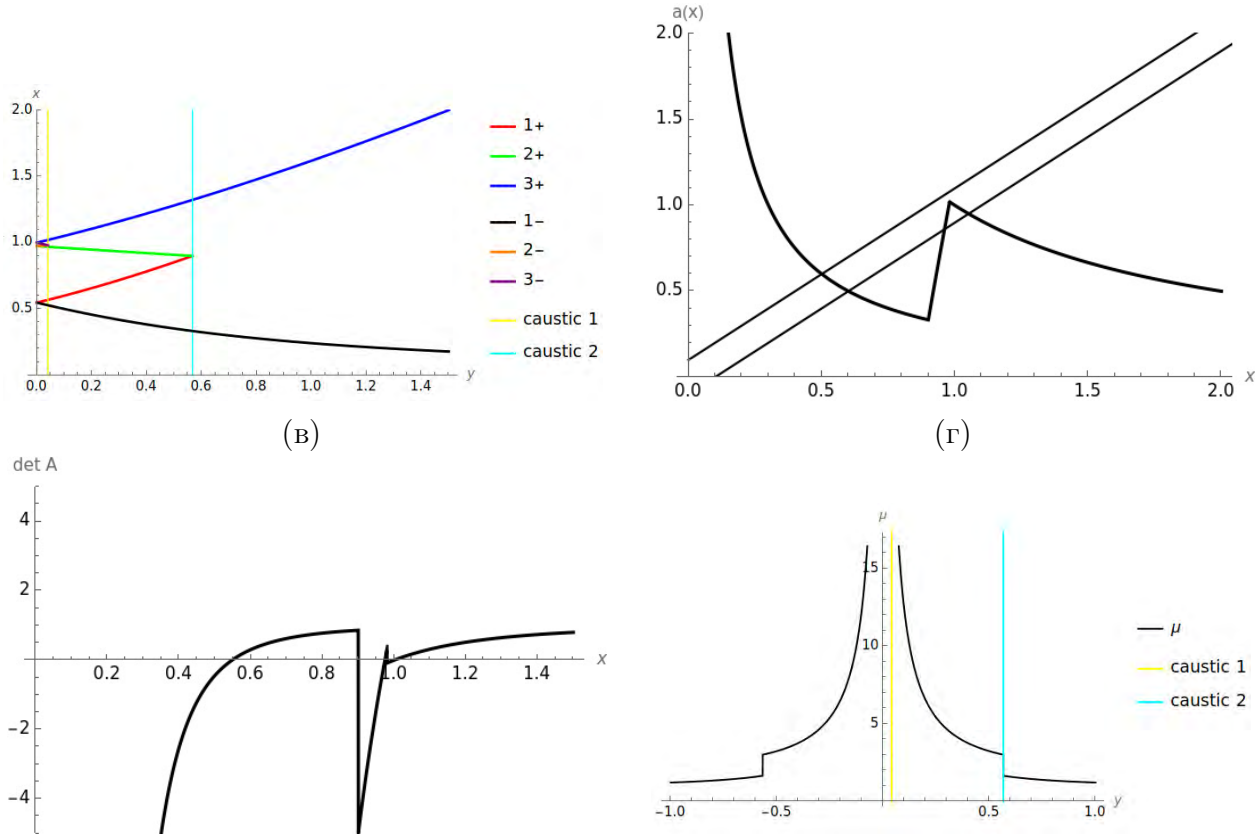


(е)

Рис. 41: Ефекти гравітаційного лінзування для системи з параметрами $m_0 = 0.3, r_1 = 0.98, r_2 = 1.2$



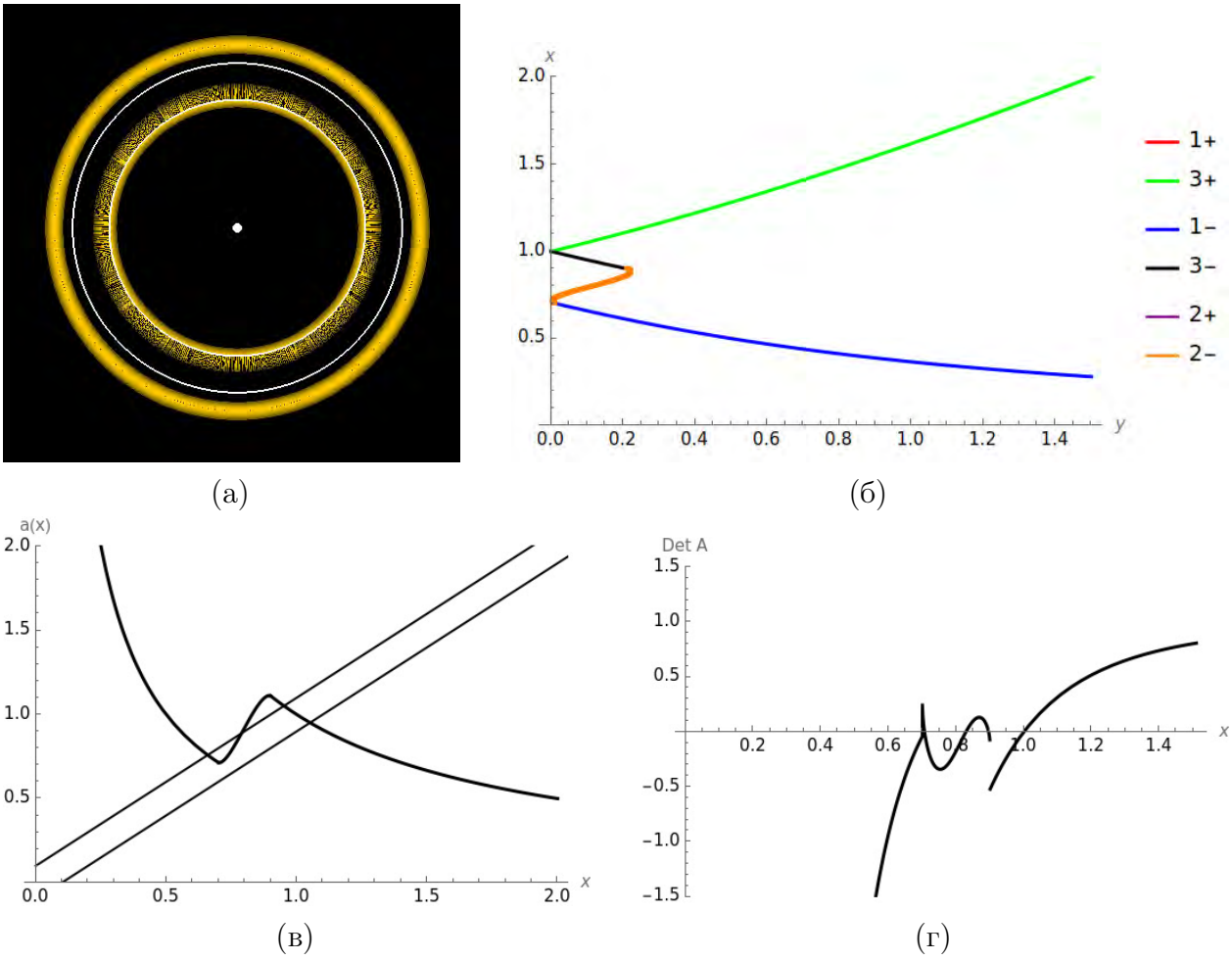
(а) (б)



(в) (г)

(д) (е)

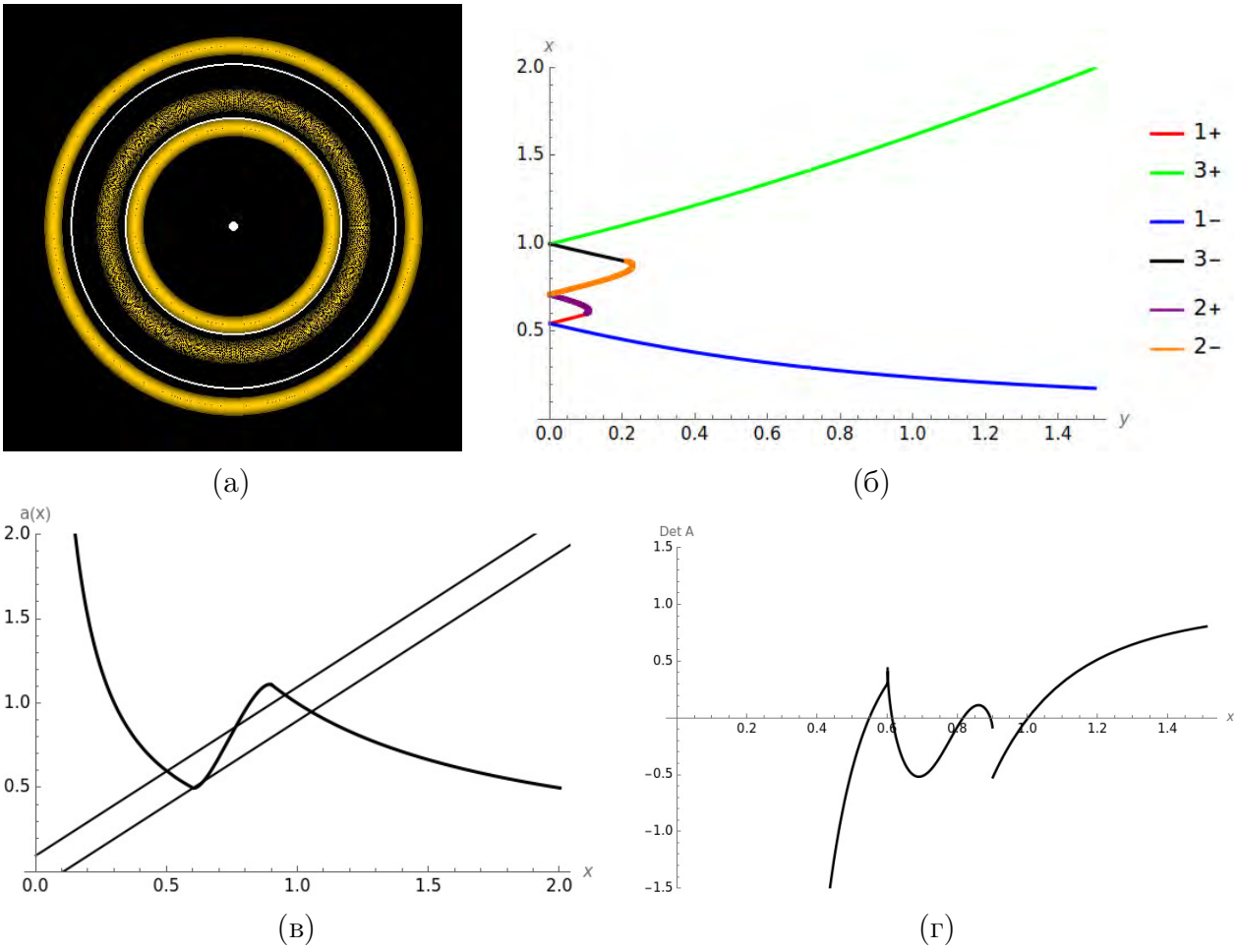
Рис. 42: Ефекти гравітаційного лінзування для системи з параметрами $m_0 = 0.3, r_1 = 0.9, r_2 = 0.98$



(а) (б)

(в) (г)

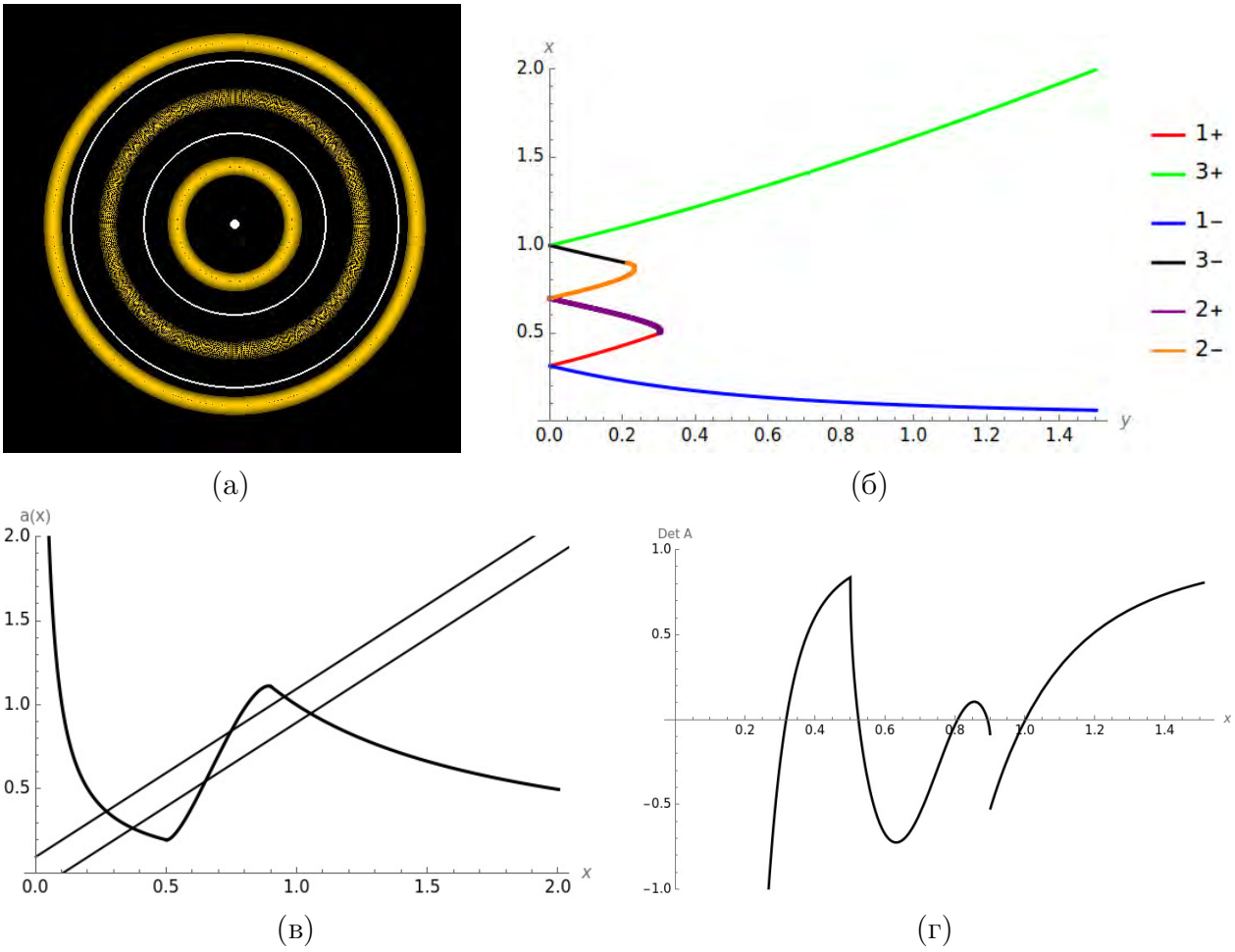
Рис. 43: Ефекти гравітаційного лінзування для системи з параметрами $m_0 = 0.5, R_{tor} = 0.8, r_0 = 0.1$



(а) (б)

(в) (г)

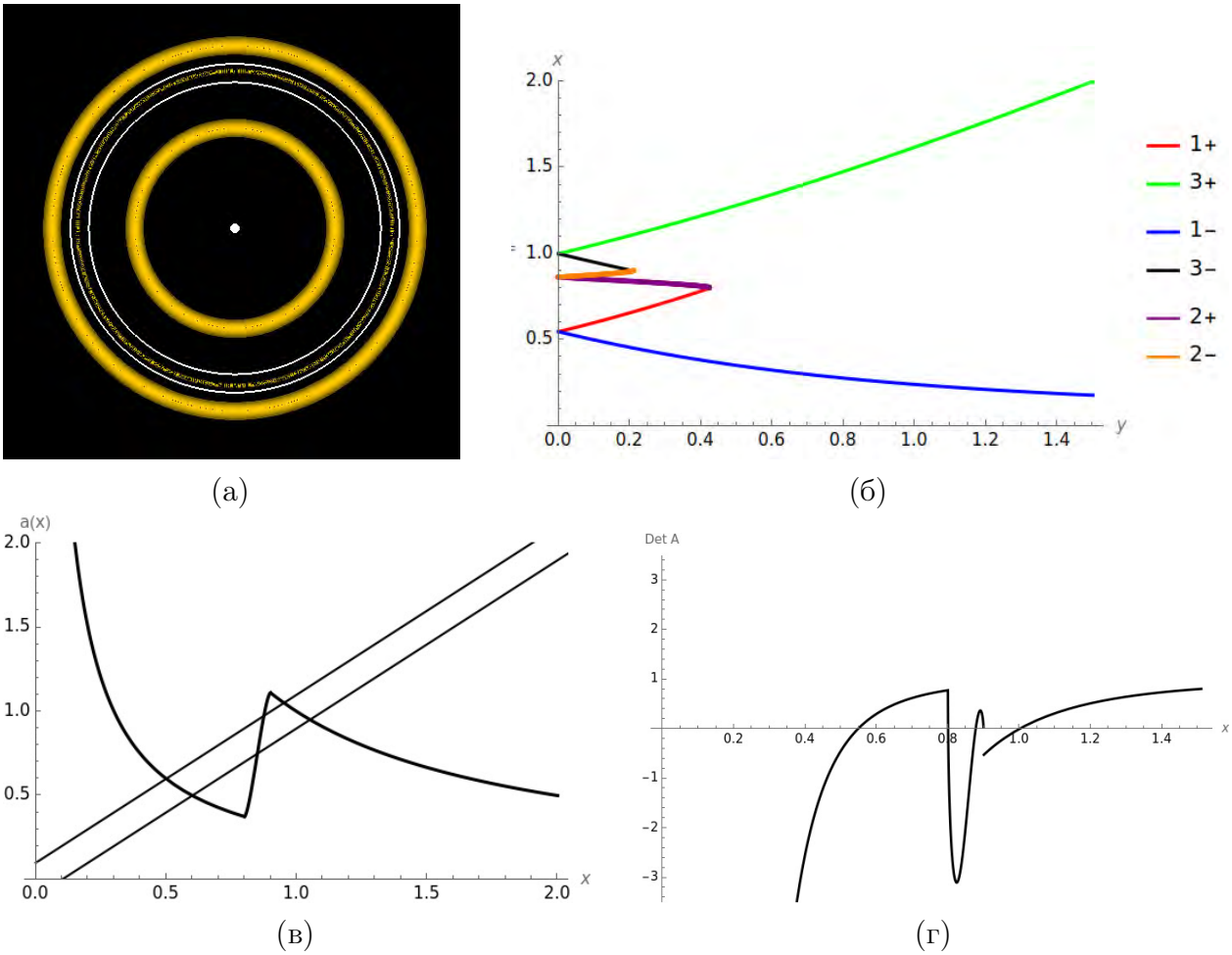
Рис. 44: Ефекти гравітаційного лінзування для системи з параметрами $m_0 = 0.3, R_{tor} = 0.75, r_0 = 0.15$



(а) (б)

(в) (г)

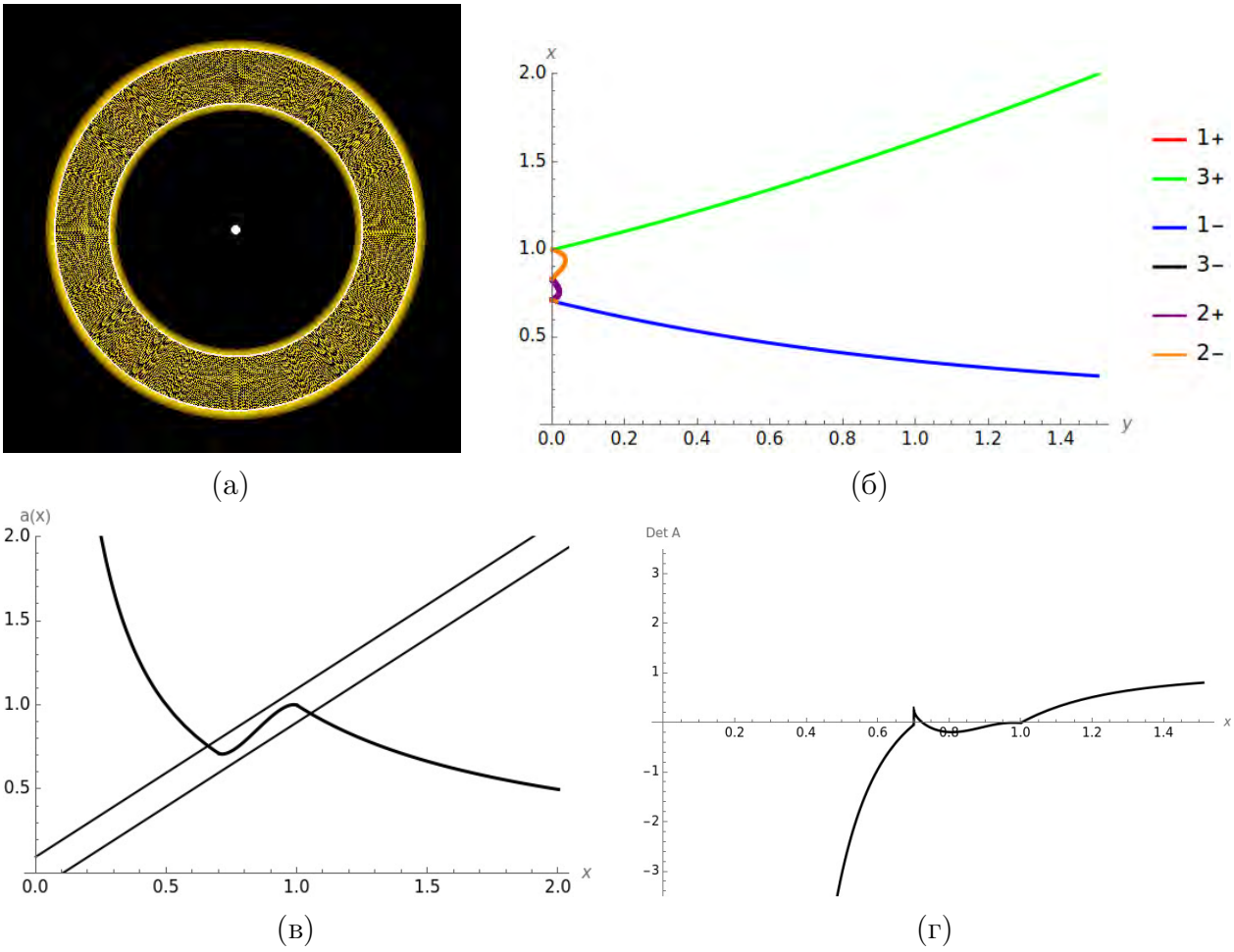
Рис. 45: Ефекти гравітаційного лінзування для системи з параметрами $m_0 = 0.1, R_{tor} = 0.7, r_0 = 0.2$



(а) (б)

(в) (г)

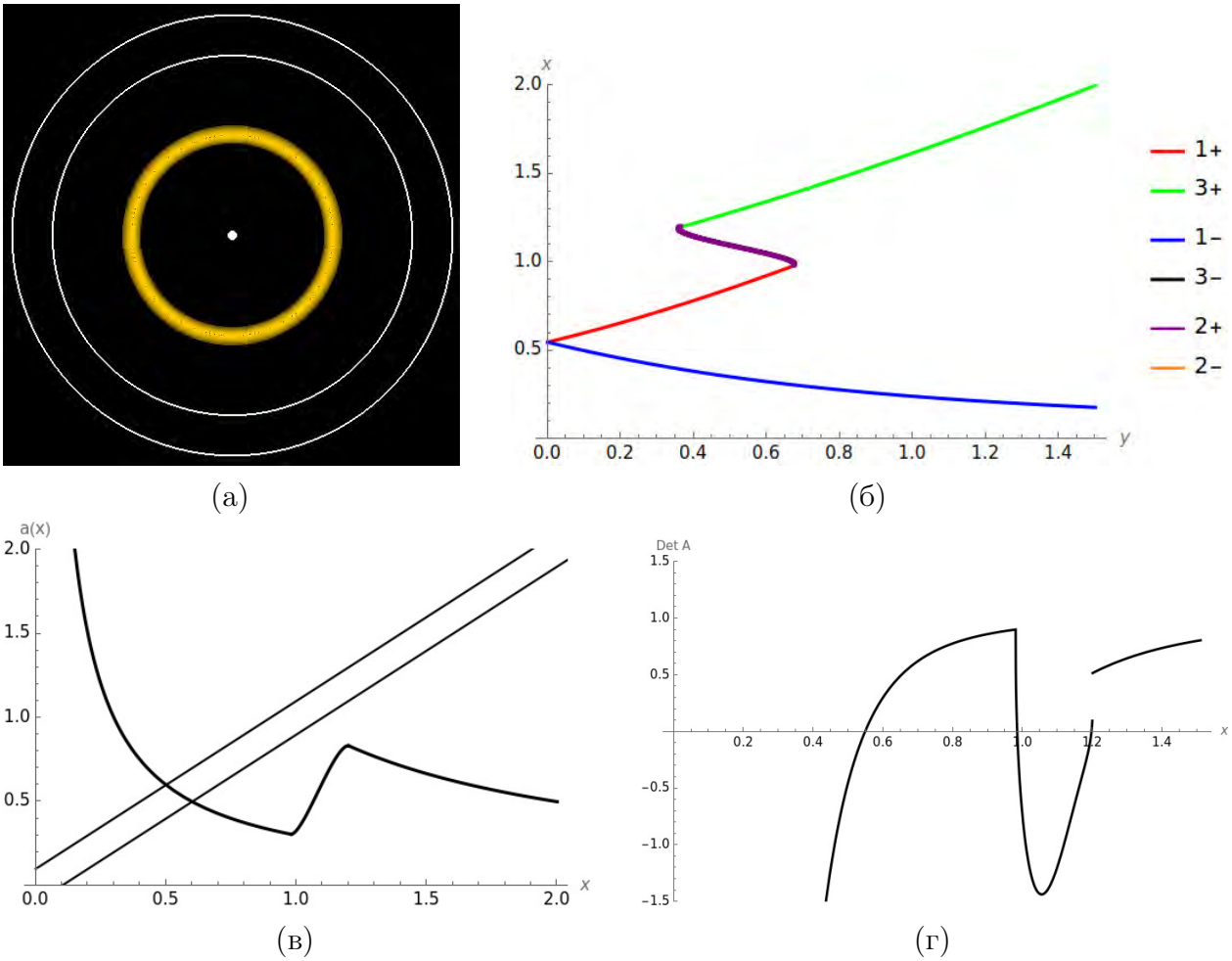
Рис. 46: Ефекти гравітаційного лінзування для системи з параметрами $m_0 = 0.3, R_{tor} = 0.85, r_0 = 0.05$



(а) (б)

(в) (г)

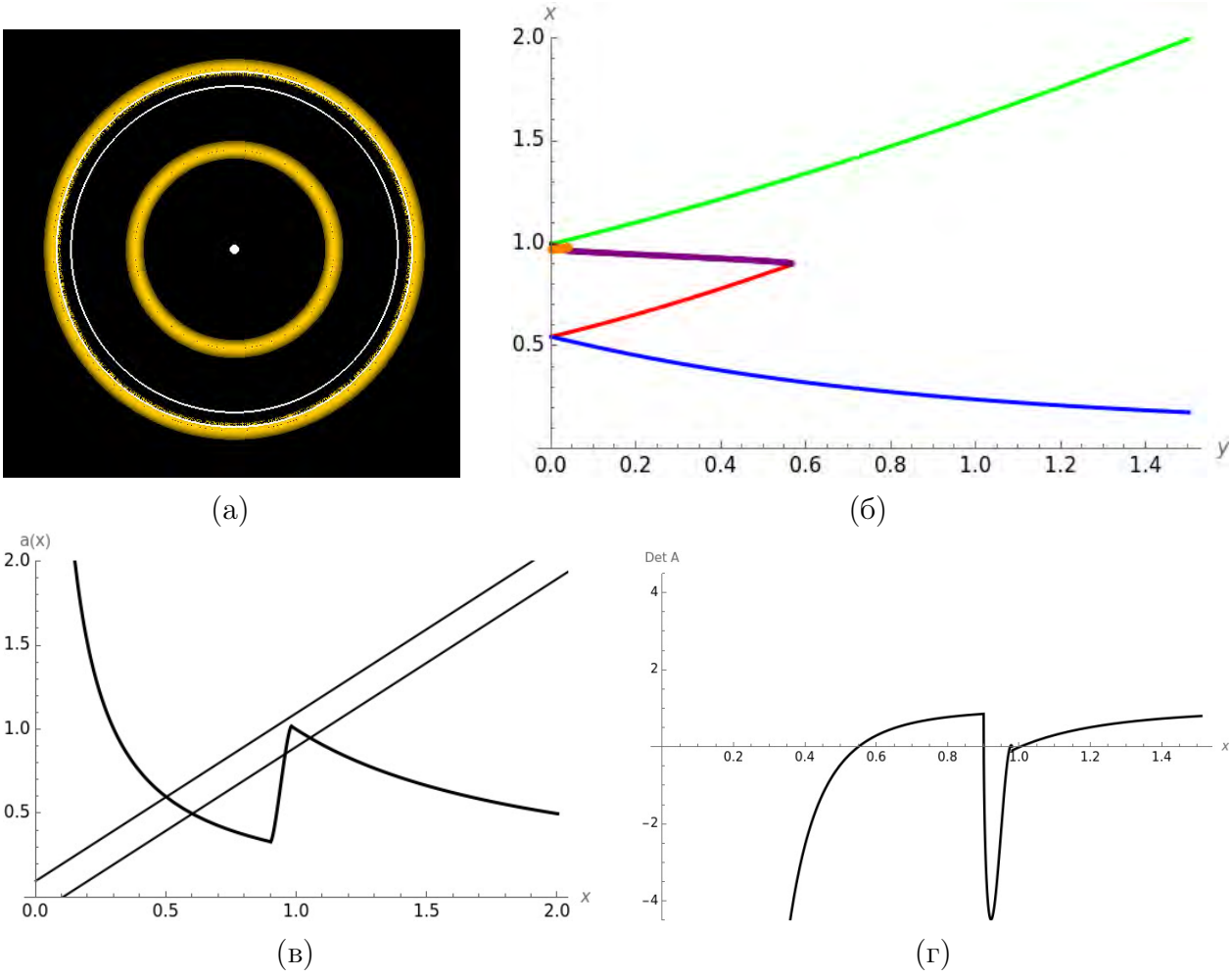
Рис. 47: Ефекти гравітаційного лінзування для системи з параметрами $m_0 = 0.3, R_{tor} = 0.85, r_0 = 0.15$



(а) (б)

(в) (г)

Рис. 48: Ефекти гравітаційного лінзування для системи з параметрами $m_0 = 0.3, R_{tor} = 1.09, r_0 = 0.11$



(а) (б)

(в) (г)

Рис. 49: Ефекти гравітаційного лінзування для системи з параметрами $m_0 = 0.3, R_{tor} = 0.94, r_0 = 0.04$

Додаток В: Код програми для побудови та візуалізації ефектів гравітаційного лінзування

```
#include <GL/glew.h>
#include <GLFW/glfw3.h>
#include <GL/glut.h>
#include <cstdio>
#include <fstream>
#include <iostream>
#include <sstream>
#include <string>
#include <string.h>
#include <vector>
#include <stdlib.h>
#include <stdio.h>
#include <random>
#include <math.h>
using namespace std;

double Pi = 3.1415927;

int radial = 200;
int tang = 200;
int NPoints = radial * tang;
double RSource = 0.1;
double Rsourse1 = 0.0;
double Distanse = 0.3;
double Distanse_im = 2;
double Scale = 0.8;
double marker = 1.3;
double step = 0;

double E0 = 2*RSource*10;
double w0 = 1*RSource*10;
double wz = 2.0*RSource*10;

double m0 = 0.1;
double md = 1 - m0;
double r1 = 0.1;
double r2 = 0.1;
```

```
38 double k = md / (r2 * r2 - r1 * r1);
39
40 double Rtor = 0.7;
41 double r0 = 0.2;
42 double k0=  (1 - m0) / (3.14159265* r0*r0 * Rtor);
43 double ray_radial = 500;
44 double ray_tang = 500;
45
46 double move_sourse=0;
47
48 double red;
49 double grean;
50 double blue;
51 double Color = 2;
52
53 void COLOR(double color)
54 {
55     int a = static_cast<int>(color);
56     switch (a)
57     {
58     case 0:
59         red = 1;
60         grean = 0.027;
61         blue = 0.027;
62         break;
63     case 1:
64         red = 1;
65         grean = 0.553;
66         blue = 0;
67         break;
68     case 2:
69         red = 1;
70         grean = 0.8;
71         blue = 0.008;
72         break;
73
74     case 3:
75         red = 0.012;
76         grean = 0.71;
77         blue = 0.11;
78         break;
```

```
79
80      case 4:
81          red = 0.063;
82          grean = 0.392;
83          blue = 0.871;
84          break;
85
86      case 5:
87          red = 0.69;
88          grean = 0.012;
89          blue = 0.71;
90          break;
91      default:
92          break;
93      }
94  }
95
96
97  static void error_callback(int error, const char* description)
98  {
99      fputs(description, stderr);
100 }
101 static void key_callback(GLFWwindow* window, int key, int scancode, int
        action, int mods)
102 {
103     if (key == GLFW_KEY_ESCAPE && action == GLFW_PRESS)
104         glfwSetWindowShouldClose(window, GL_TRUE);
105 }
106
107 class Sourse_Array {
108 public:
109     class Sourse
110     {
111
112     public:
113
114         double** a;
115         int nu;
116
117         Sourse()
118         {
```

```
119             this->nu = NPoints;
120             a = new double* [nu];
121             for (int i = 0; i < nu; i++)
122             {
123                 a[i] = new double[4];
124             }
125 
126             for (int i = 0; i < nu; i++)
127             {
128                 a[i][0] = 0;
129                 a[i][1] = 0;
130                 a[i][2] = 0;
131                 a[i][3] = 0;
132             }
133 
134         }
135 
136         Sourse(double x, double y)
137         {
138             this->nu = NPoints;
139             a = new double* [nu];
140             for (int i = 0; i < nu; i++)
141             {
142                 a[i] = new double[4];
143             }
144 
145             for (int i = 0; i < nu; i++)
146             {
147                 double fi = static_cast <double> (rand()) / (static_cast
    <double> (RAND_MAX / (3.14159 * 2)));
148                 double r = static_cast <double> (rand()) / (static_cast
    <double> (RAND_MAX / RSource));
149                 a[i][0] = x + r * cos(fi);
150                 a[i][1] = y + r * sin(fi);
151                 a[i][2] = E0 * w0 / wz * pow(2.71828, (-pow(r / RSource, 2.0)
    / pow(wz, 2.0)));
152                 a[i][3] = Color;
153             }
154         }
155 
156         void Central_Mass(const Sourse& other)
```

```
157         {
158             this->nu = 2 * NPoints;
159             a = new double* [nu];
160             for (int i = 0; i < nu; i++)
161             {
162                 a[i] = new double[4];
163             }
164 
165             for (int i = 0; i < NPoints; i++)
166             {
167                 for (int j = 0; j < 3; j++)
168                 {
169                     double y = sqrt(other.a[i][0] * other.a[i][0] +
    other.a[i][1] * other.a[i][1]);
170                     double x1, x2;
171 
172                     this->a[i * 2][0] = other.a[i][0] * 1.0 / 2.0 * (y +
    sqrt(4.0 + y * y)) / y;
173                     this->a[i * 2][1] = other.a[i][1] * 1.0 / 2.0 * (y +
    sqrt(4.0 + y * y)) / y;
174                     this->a[i * 2 + 1][0] = other.a[i][0] * 1.0 / 2.0 * (y -
    sqrt(4.0 + y * y)) / y;
175                     this->a[i * 2 + 1][1] = other.a[i][1] * 1.0 / 2.0 * (y -
    sqrt(4.0 + y * y)) / y;
176                     x1 = sqrt(this->a[i * 2][0] * this->a[i * 2][0] +
    this->a[i * 2][1] * this->a[i * 2][1]);
177                     x2 = sqrt(this->a[i * 2 + 1][0] * this->a[i * 2 + 1][0] +
    this->a[i * 2 + 1][1] * this->a[i * 2 + 1][1]);
178                     this->a[i * 2][2] = other.a[i][2];
179                     this->a[i * 2 + 1][2] = other.a[i][2];
180                     this->a[i * 2][3] = other.a[i][3];
181                     this->a[i * 2 + 1][3] = other.a[i][3];
182                 }
183             }
184 
185         }
186 
187         void Disk_With_Central_Mass(const Sourse& other)
188         {
189             this->nu = 6 * NPoints;
190             a = new double* [nu];
```

```
191             for (int i = 0; i < nu; i++)
192             {
193                 a[i] = new double[4];
194             }
195 
196             for (int i = 0; i < NPoints; i++)
197             {
198 
199                 double y = sqrt(other.a[i][0] * other.a[i][0] +
    other.a[i][1] * other.a[i][1]);
200                 double x1, x2, x3, x4, x5, x6;
201 
202                 this->a[i * 6][0] = other.a[i][0] * 1.0 / 2.0 * (y +
    sqrt(4.0 + y * y)) / y;
203                 this->a[i * 6][1] = other.a[i][1] * 1.0 / 2.0 * (y +
    sqrt(4.0 + y * y)) / y;
204                 this->a[i * 6 + 1][0] = other.a[i][0] * 1.0 / 2.0 * (y -
    sqrt(4.0 + y * y)) / y;
205                 this->a[i * 6 + 1][1] = other.a[i][1] * 1.0 / 2.0 * (y -
    sqrt(4.0 + y * y)) / y;
206                 x1 = sqrt(this->a[i * 6][0] * this->a[i * 6][0] +
    this->a[i * 6][1] * this->a[i * 6][1]);
207                 x2 = sqrt(this->a[i * 6 + 1][0] * this->a[i * 6 + 1][0] +
    this->a[i * 6 + 1][1] * this->a[i * 6 + 1][1]);
208 
209                 this->a[i * 6 + 2][0] = other.a[i][0] * 1.0 / 2.0 * (y +
    sqrt(4.0 * m0 + y * y)) / y;
210                 this->a[i * 6 + 2][1] = other.a[i][1] * 1.0 / 2.0 * (y +
    sqrt(4.0 * m0 + y * y)) / y;
211                 this->a[i * 6 + 3][0] = other.a[i][0] * 1.0 / 2.0 * (y -
    sqrt(4.0 * m0 + y * y)) / y;
212                 this->a[i * 6 + 3][1] = other.a[i][1] * 1.0 / 2.0 * (y -
    sqrt(4.0 * m0 + y * y)) / y;
213                 x3 = sqrt(this->a[i * 6 + 2][0] * this->a[i * 6 + 2][0] +
    this->a[i * 6 + 2][1] * this->a[i * 6 + 2][1]);
214                 x4 = sqrt(this->a[i * 6 + 3][0] * this->a[i * 6 + 3][0] +
    this->a[i * 6 + 3][1] * this->a[i * 6 + 3][1]);
215 
216 
217                 this->a[i * 6 + 4][0] = other.a[i][0] * (-y + sqrt(-4.0 *
    (-1 + k) * (m0 - k * r1 * r1) + y * y)) / (-2 + 2 * k) / y;
```

```
218                     this->a[i * 6 + 4][1] = other.a[i][1] * (-y + sqrt(-4.0 *
    (-1 + k) * (m0 - k * r1 * r1) + y * y)) / (-2 + 2 * k) / y;
219                     this->a[i * 6 + 5][0] = other.a[i][0] * (-y - sqrt(-4.0 *
    (-1 + k) * (m0 - k * r1 * r1) + y * y)) / (-2 + 2 * k) / y;
220                     this->a[i * 6 + 5][1] = other.a[i][1] * (-y - sqrt(-4.0 *
    (-1 + k) * (m0 - k * r1 * r1) + y * y)) / (-2 + 2 * k) / y;
221                     x5 = sqrt(this->a[i * 6 + 4][0] * this->a[i * 6 + 4][0] +
    this->a[i * 6 + 4][1] * this->a[i * 6 + 4][1]);
222                     x6 = sqrt(this->a[i * 6 + 5][0] * this->a[i * 6 + 5][0] +
    this->a[i * 6 + 5][1] * this->a[i * 6 + 5][1]);
223
224                     this->a[i * 6][3] = other.a[i][3];
225                     this->a[i * 6 + 1][3] = other.a[i][3];
226                     this->a[i * 6 + 2][3] = other.a[i][3];
227                     this->a[i * 6 + 3][3] = other.a[i][3];
228                     this->a[i * 6 + 4][3] = other.a[i][3];
229                     this->a[i * 6 + 5][3] = other.a[i][3];
230
231
232                     if (x1 >= r2) {
233                         this->a[i * 6][2] = other.a[i][2];
234                     }
235                     else {
236                         this->a[i * 6][2] = 0;
237                     }
238
239                     if (x2 >= r2) {
240                         this->a[i * 6 + 1][2] = other.a[i][2];
241                     }
242                     else {
243                         this->a[i * 6 + 1][2] = 0;
244                     }
245
246                     if (x3 <= r1) {
247                         this->a[i * 6 + 2][2] = other.a[i][2];
248                     }
249                     else {
250                         this->a[i * 6 + 2][2] = 0;
251                     }
252
253                     if (x4 <= r1) {
```

```
                    this->a[i * 6 + 3][2] = other.a[i][2];
                }
                else {
                    this->a[i * 6 + 3][2] = 0;
                }

                if (x5 <= r2 && x5 >= r1) {
                    this->a[i * 6 + 4][2] = other.a[i][2];
                }
                else {
                    this->a[i * 6 + 4][2] = 0;
                }

                if (x6 <= r2 && x6 >= r1) {
                    this->a[i * 6 + 5][2] = other.a[i][2];
                }
                else {
                    this->a[i * 6 + 5][2] = 0;
                }
            }
        }

        void Tor_With_Central_Mass(const Sourse& other)
        {
            int n_image = 4;
            this->nu = n_image * NPoints;
            a = new double* [nu];
            for (int i = 0; i < nu; i++)
            {
                a[i] = new double[4];
            }

            for (int i = 0; i < NPoints; i++)
            {
                    double y = sqrt(other.a[i][0] * other.a[i][0] +
other.a[i][1] * other.a[i][1]);
                    double x1, x2, x3, x4;

                    this->a[i * n_image][0] = other.a[i][0] * 1.0 / 2.0 * (y +
sqrt(4.0 * m0 + y * y)) / y;
                    this->a[i * n_image][1] = other.a[i][1] * 1.0 / 2.0 * (y +
```

```
sqrt(4.0 * m0 + y * y)) / y;
293                 this->a[i * n_image + 1][0] = other.a[i][0] * 1.0 / 2.0 *
(y - sqrt(4.0 * m0 + y * y)) / y;
294                 this->a[i * n_image + 1][1] = other.a[i][1] * 1.0 / 2.0 *
(y - sqrt(4.0 * m0 + y * y)) / y;
295                 x1 = sqrt(this->a[i * n_image][0] * this->a[i *
n_image][0] + this->a[i * n_image][1] * this->a[i * n_image][1]);
296                 x2 = sqrt(this->a[i * n_image + 1][0] * this->a[i *
n_image + 1][0] + this->a[i * n_image + 1][1] * this->a[i * n_image +
1][1]);
297
298                 this->a[i * n_image + 2][0] = other.a[i][0] * 1.0 / 2.0 *
(y + sqrt(4.0  + y * y)) / y;
299                 this->a[i * n_image + 2][1] = other.a[i][1] * 1.0 / 2.0 *
(y + sqrt(4.0  + y * y)) / y;
300                 this->a[i * n_image + 3][0] = other.a[i][0] * 1.0 / 2.0 *
(y - sqrt(4.0  + y * y)) / y;
301                 this->a[i * n_image + 3][1] = other.a[i][1] * 1.0 / 2.0 *
(y - sqrt(4.0  + y * y)) / y;
302                 x3 = sqrt(this->a[i * n_image + 2][0] * this->a[i *
n_image + 2][0] + this->a[i * n_image + 2][1] * this->a[i * n_image +
2][1]);
303                 x4 = sqrt(this->a[i * n_image + 3][0] * this->a[i *
n_image + 3][0] + this->a[i * n_image + 3][1] * this->a[i * n_image +
3][1]);
304
305                 this->a[i * n_image][3] = other.a[i][3];
306                 this->a[i * n_image + 1][3] = other.a[i][3];
307                 this->a[i * n_image + 2][3] = other.a[i][3];
308                 this->a[i * n_image + 3][3] = other.a[i][3];
309
310                 if (x1 <= Rtor-r0) {
311                     this->a[i * n_image][2] = other.a[i][2];
312                 }
313                 else {
314                     this->a[i * n_image][2] = 0;
315                 }
316
317                 if (x2 <= Rtor - r0) {
318                     this->a[i * n_image + 1][2] = other.a[i][2];
319                 }
```

```
320                     else {
321                         this->a[i * n_image + 1][2] = 0;
322                     }
323 
324                     if (x3 >=Rtor+ r0) {
325                         this->a[i * n_image + 2][2] = other.a[i][2];
326                     }
327                     else {
328                         this->a[i * n_image + 2][2] = 0;
329                     }
330 
331                     if (x4 >=Rtor+ r0) {
332                         this->a[i * n_image + 3][2] = other.a[i][2];
333                     }
334                     else {
335                         this->a[i * n_image + 3][2] = 0;
336                     }
337             }
338         }
339 
340         void by_Cord(double x, double y,int sourse_type)
341         {
342 
343             for (int i = 0; i < tang; i++)
344             {
345                 double fi = 0 + 2 * 3.3 / tang * i;
346                 for (int j = 0; j < radial; j++) {
347                     double r = RSource * (j + 1) / (radial + 1);
348                     this->a[i * radial + j][0] = x + move_sourse + r * cos(fi)
    + step;
349                     this->a[i * radial + j][1] = y + r * sin(fi);
350                     if (sourse_type == 1) {
351                         this->a[i * radial + j][2] = E0 * w0 / wz *
    pow(2.71828, (-pow(r * 20, 2.0) / pow(wz, 2.0)));
352                     }
353                     if (sourse_type == 2) {
354                         if (cos(fi) > 0) {
355                             this->a[i * radial + j][2] = E0 * w0 / wz *
    pow(2.71828, (-pow(r * 20, 2.0) / pow(wz, 2.0)));
356                         }
357                         else {
```

```
                        this->a[i * radial + j][2] = 0;
                    }


                }
                if (sourse_type == 3) {
                    if (cos(fi) > 0&& sin(fi) > 0) {
                        this->a[i * radial + j][2] = E0 * w0 / wz *
pow(2.71828, (-pow(r * 20, 2.0) / pow(wz, 2.0)));
                    }
                    else {
                        this->a[i * radial + j][2] = 0;
                    }
                }
                if (sourse_type == 4) {
                    if (r> Rsourse1) {
                        this->a[i * radial + j][2] = E0 * w0 / wz *
pow(2.71828, (-pow(r * 20, 2.0) / pow(wz, 2.0)));
                    }
                    else {
                        this->a[i * radial + j][2] = 0;
                    }


                }
                this->a[i * radial + j][3] = Color;
            }

        }

    }

    ~Sourse()
    {
        if (this->a != nullptr) {
            for (int i = 0; i < this->nu; i++)
            {
                delete[] this->a[i];
                this->a[i] = nullptr;
            }
            delete[] this->a;
```

```
397                 this->a = nullptr;
398             }
399         }
400 
401         void Print_Array(int n)
402         {
403             for (int i = 0; i < n; i++)
404             {
405                 for (int j = 0; j < 1; j++)
406                 {
407                     cout << a[i][2] << "\t" << endl;
408                 }
409                 cout << endl;
410             }
411         }
412     };
413 
414     Sourse** sourse;
415     Sourse** image;
416     int n;
417     int lens_type;
418     int sourse_type;
419     Sourse_Array(int lens_type, int n,int sourse_type)
420 
421     {
422         this->n = n;
423         this->lens_type = lens_type;
424         this->sourse_type = sourse_type;
425         sourse = new Sourse * [n];
426         for (int i = 0; i < n; i++)
427         {
428             sourse[i] = new Sourse[n];
429 
430         }
431 
432         image = new Sourse * [n];
433         for (int i = 0; i < n; i++)
434         {
435             image[i] = new Sourse[n];
436         }
437 
```

```
438         for (int i = 0; i < n; i++)
439         {
440             for (int j = 0; j < n; j++)
441             {
442                 double X_center = -Distanse * (static_cast <double>(n) / 2.0 +
    -0.5) + Distanse * static_cast <double>(j);
443                 double Y_center = -Distanse * (static_cast <double>(n) / 2.0 -
    0.5) + Distanse * static_cast <double>(i);
444                 sourse[i][j].by_Cord(X_center, Y_center,this->sourse_type);
445 
446                 if (lens_type == 1) {
447                     image[i][j].Central_Mass(sourse[i][j]);
448                 }
449 
450                 if (lens_type == 2) {
451                     image[i][j].Disk_With_Central_Mass(sourse[i][j]);
452                 }
453 
454 
455                 if (lens_type == 3) {
456                     image[i][j].Tor_With_Central_Mass(sourse[i][j]);
457                 }
458             }
459         }
460     }
461     ~Sourse_Array()
462     {
463         for (int l = 0; l < this->n; l++)
464         {
465             for (int o = 0; o < this->n; o++) {
466 
467                 sourse[l][o].~Sourse();
468                 image[l][o].~Sourse();
469             }
470 
471             delete[] this->sourse[l];
472             delete[] this->image[l];
473             this->sourse[l] = nullptr;
474             this->image[l] = nullptr;
475         }
476         delete[] this->sourse;
```

```
477         delete[] this->image;
478         this->sourse = nullptr;
479         this->image = nullptr;
480 
481     }
482 
483     void Show_Sourse() {
484         double x, y;
485         for (int i = 0; i < n; i++)
486         {
487             for (int j = 0; j < n; j++)
488             {
489 
490                 for (int d = 0; d < radial - 1; d++)
491                 {
492                     for (int k = 0; k < tang - 1; k++)
493                     {
494 
495                         glBegin(GL_TRIANGLE_STRIP);
496                         x = this->sourse[i][j].a[k * radial + d][0];
497                         y = this->sourse[i][j].a[k * radial + d][1];
498 
499                         COLOR(this->sourse[i][j].a[k * radial + d][3]);
500                         glColor3f(red * this->sourse[i][j].a[k * radial +
    d][2], grean * this->sourse[i][j].a[k * radial + d][2], blue *
    this->sourse[i][j].a[k * radial + d][2]);
501 
502                         glVertex3f(
503                             (x), y, 0);
504 
505 
506                         x = this->sourse[i][j].a[k * radial + d + 1][0];
507                         y = this->sourse[i][j].a[k * radial + d + 1][1];
508                         COLOR(this->sourse[i][j].a[k * radial + d + 1][3]);
509                         glColor3f(red * this->sourse[i][j].a[k * radial + d +
    1][2], grean * this->sourse[i][j].a[k * radial + d + 1][2], blue *
    this->sourse[i][j].a[k * radial + d + 1][2]);
510                         glVertex3f(x, y, 0);
511 
512 
513 
```

```
514                         x = this->sourse[i][j].a[(k + 1) * radial + d][0];
515                         y = this->sourse[i][j].a[(k + 1) * radial + d][1];
516                         COLOR(this->sourse[i][j].a[(k + 1) * radial + d][3]);
517                         glColor3f(red * this->sourse[i][j].a[(k + 1) * radial
    + d][2], grean * this->sourse[i][j].a[(k + 1) * radial + d][2], blue *
    this->sourse[i][j].a[(k + 1) * radial + d][2]);
518                         glVertex3f(x, y, 0);
519
520
521                         x = this->sourse[i][j].a[(k + 1) * radial + d + 1][0];
522                         y = this->sourse[i][j].a[(k + 1) * radial + d + 1][1];
523                         COLOR(this->sourse[i][j].a[(k + 1) * radial + d +
    1][3]);
524                         glColor3f(red * this->sourse[i][j].a[(k + 1) * radial
    + d + 1][2], grean * this->sourse[i][j].a[(k + 1) * radial + d + 1][2],
    blue * this->sourse[i][j].a[(k + 1) * radial + d + 1][2]);
525
526                         glVertex3f(x, y, 0);
527
528                         glEnd(); //END
529                     }
530                 }
531
532
533                 glBegin(GL_TRIANGLE_FAN);
534                 for (int u = 0; u < tang; u++)
535                 {
536                     x = this->sourse[i][j].a[u * radial][0];
537                     y = this->sourse[i][j].a[u * radial][1];
538
539                     if (this->sourse[i][j].a[u * radial][2] > 0) {
540                         COLOR(this->sourse[i][j].a[u * radial][3]);
541                         glColor3f(red * this->sourse[i][j].a[u * radial][2],
    grean * this->sourse[i][j].a[u * radial][2], blue * this->sourse[i][j].a[u
    * radial][2]);
542
543                         glVertex3f(x, y, 0);
544
545                     }
546                 }
547                 glEnd();
```

```
548
549             }
550         }
551     }
552
553     void ray_trasing_for_tor()
554     {
555
556         for (int ii = 0; ii < n; ii++)
557         {
558             for (int jj = 0; jj < n; jj++)
559             {
560                 double X_center = -Distanse * (static_cast <double>(n) / 2.0 +
    -0.5) + Distanse * static_cast <double>(jj);
561                 double Y_center = -Distanse * (static_cast <double>(n) / 2.0 -
    0.5) + Distanse * static_cast <double>(ii);
562
563                 COLOR(this->sourse[ii][jj].a[0][3]);
564
565                 for (int i = 0; i < ray_radial; i++)
566                 {
567                     for (int j = 0; j < ray_tang; j++)
568                     {
569                         glBegin(GL_TRIANGLE_STRIP);
570                         double x = (Rtor - r0 + 2 * r0 / ray_radial * i) *
    cos(2 * 3.15159265 / ray_tang * j);
571                         double y = (Rtor - r0 + 2 * r0 / ray_radial * i) *
    sin(2 * 3.15159265 / ray_tang * j);
572                         double X = sqrt(x * x + y * y);
573                         double zeta =asin(X/Rtor-r0/Rtor);
574                         double
    I2=r0*Rtor/4*(2*zeta+sin(2*zeta)+3.14159)-r0*r0/3*pow(cos(zeta),3);
575                         double X_sourse = (X - m0 / X - 2*k0 *I2)* x / X;
576                         double Y_sourse = (X - m0 / X - 2*k0 *I2)*y / X;
577
578                         double hypotetical_radius_x = abs(X_sourse - X_center);
579                         double hypotetical_radius_y = abs(Y_sourse - Y_center);
580                         double hypotetical_radius =
    sqrt(pow(hypotetical_radius_x,2)+ pow(hypotetical_radius_y, 2));
581                         bool due_to_sourse_type=false;
582                         if (sourse_type == 1)
```

```
583                         {
584                             due_to_sourse_type = true;
585                             Rsourse1 = 0;
586                         }
587                         if (sourse_type == 2)
588                         {
589                             if (X_sourse > 0) {
590                                 due_to_sourse_type = true;
591                             }
592                             Rsourse1 = 0;
593 
594                         }
595                         if (sourse_type == 3)
596                         {
597                             if (X_sourse > 0&&Y_sourse>0) {
598                                 due_to_sourse_type = true;
599                             }
600                             Rsourse1 = 0;
601 
602                         }
603                         if (sourse_type == 4)
604                         {
605                             if (sqrt(pow(X_sourse, 2) + pow(Y_sourse,
    2))>Rsourse1) {
606                                 due_to_sourse_type = true;
607                             }
608 
609                         }
610                         if ( hypotetical_radius < RSource )
611                         {
612                             double I= E0 * w0 / wz * pow(2.71828,
    (-pow(hypotetical_radius * 20, 2.0) / pow(wz, 2.0)));
613 
614                             glColor3f(red * I, grean * I, blue*I);
615                             glVertex3f(x, y, 0);
616 
617                         }
618                         else  if (hypotetical_radius >RSource)
619                         {
620                             double I = E0 * w0 / wz * pow(2.71828,
    (-pow(hypotetical_radius * 20, 2.0) / pow(wz, 2.0)));
```

```
621
622                         glColor3f(1, 0, 0);
623
624                     }
625                     x = (Rtor - r0 + 2 * r0 / ray_radial * (i+1)) * cos(2
    * 3.15159265 / ray_tang * j);
626                     y = (Rtor - r0 + 2 * r0 / ray_radial * (i+1)) * sin(2
    * 3.15159265 / ray_tang * j);
627                     X = sqrt(x * x + y * y);
628                     X_sourse = (X - m0 / X + k0 / (3 * X) * (sqrt(pow(r0,
    2) - pow(X - Rtor, 2)) * (2 * pow(r0, 2) + (Rtor - X) * (Rtor + 2 * X)) + 3
    * pow(r0, 2) * Rtor * (atan((Rtor - X) / sqrt(pow(r0, 2) - pow((X - Rtor),
    2))) - 3.14159265 / 2))) * x / X;
629                     Y_sourse = (X - m0 / X + k0 / (3 * X) * (sqrt(pow(r0,
    2) - pow(X - Rtor, 2)) * (2 * pow(r0, 2) + (Rtor - X) * (Rtor + 2 * X)) + 3
    * pow(r0, 2) * Rtor * (atan((Rtor - X) / sqrt(pow(r0, 2) - pow((X - Rtor),
    2))) - 3.14159265 / 2))) * y / X;
630
631                     hypotetical_radius_x = abs(X_sourse - X_center);
632                     hypotetical_radius_y = abs(Y_sourse - Y_center);
633                     hypotetical_radius = sqrt(pow(hypotetical_radius_x, 2)
    + pow(hypotetical_radius_y, 2));
634
635                     if ( hypotetical_radius < RSource )
636                     {
637                         double I = E0 * w0 / wz * pow(2.71828,
    (-pow(hypotetical_radius * 20, 2.0) / pow(wz, 2.0)));
638                         glColor3f(red * I, grean * I, blue * I);
639                         glVertex3f(x, y, 0);
640                     }
641
642                     x = (Rtor - r0 + 2 * r0 / ray_radial * (i )) * cos(2 *
    3.15159265 / ray_tang * (j+1));
643                     y = (Rtor - r0 + 2 * r0 / ray_radial * (i)) * sin(2 *
    3.15159265 / ray_tang * (j+1));
644                     X = sqrt(x * x + y * y);
645                     X_sourse = (X - m0 / X + k0 / (3 * X) * (sqrt(pow(r0,
    2) - pow(X - Rtor, 2)) * (2 * pow(r0, 2) + (Rtor - X) * (Rtor + 2 * X)) + 3
    * pow(r0, 2) * Rtor * (atan((Rtor - X) / sqrt(pow(r0, 2) - pow((X - Rtor),
    2))) - 3.14159265 / 2))) * x / X;
646                     Y_sourse = (X - m0 / X + k0 / (3 * X) * (sqrt(pow(r0,
```

```
2) - pow(X - Rtor, 2)) * (2 * pow(r0, 2) + (Rtor - X) * (Rtor + 2 * X)) + 3
* pow(r0, 2) * Rtor * (atan((Rtor - X) / sqrt(pow(r0, 2) - pow((X - Rtor),
2))) - 3.14159265 / 2)))* y / X;
647
648                     hypotetical_radius_x = abs(X_sourse - X_center);
649                     hypotetical_radius_y = abs(Y_sourse - Y_center);
650                     hypotetical_radius = sqrt(pow(hypotetical_radius_x, 2)
+ pow(hypotetical_radius_y, 2));
651
652                     if ( hypotetical_radius < RSource )
653                     {
654
655                         double I = E0 * w0 / wz * pow(2.71828,
(-pow(hypotetical_radius * 20, 2.0) / pow(wz, 2.0)));
656                         glColor3f(red * I, grean * I, blue * I);
657                         glVertex3f(x, y, 0);
658                     }
659
660                     x = (Rtor - r0 + 2 * r0 / ray_radial * (i+1)) * cos(2
* 3.15159265 / ray_tang * (j + 1));
661                     y = (Rtor - r0 + 2 * r0 / ray_radial * (i+1)) * sin(2
* 3.15159265 / ray_tang * (j + 1));
662                     X = sqrt(x * x + y * y);
663                     X_sourse = (X - m0 / X + k0 / (3 * X) * (sqrt(pow(r0,
2) - pow(X - Rtor, 2)) * (2 * pow(r0, 2) + (Rtor - X) * (Rtor + 2 * X)) + 3
* pow(r0, 2) * Rtor * (atan((Rtor - X) / sqrt(pow(r0, 2) - pow((X - Rtor),
2))) - 3.14159265 / 2))) * x / X;
664                     Y_sourse = (X - m0 / X + k0 / (3 * X) * (sqrt(pow(r0,
2) - pow(X - Rtor, 2)) * (2 * pow(r0, 2) + (Rtor - X) * (Rtor + 2 * X)) + 3
* pow(r0, 2) * Rtor * (atan((Rtor - X) / sqrt(pow(r0, 2) - pow((X - Rtor),
2))) - 3.14159265 / 2)))* y / X;
665
666                     hypotetical_radius_x = abs(X_sourse - X_center);
667                     hypotetical_radius_y = abs(Y_sourse - Y_center);
668                     hypotetical_radius = sqrt(pow(hypotetical_radius_x, 2)
+ pow(hypotetical_radius_y, 2));
669
670                     if ( hypotetical_radius < RSource )
671                     {
672                         double I = E0 * w0 / wz * pow(2.71828,
(-pow(hypotetical_radius * 20, 2.0) / pow(wz, 2.0)));
```

```
673                             glColor3f(red * I, grean * I, blue * I);
674                             glVertex3f(x, y, 0);
675                         }
676                         glEnd();
677                     }
678                 }
679             }
680         }
681     }
682
683     void Show_Images()
684     {
685         double x, y;
686         for (int i = 0; i < n; i++)
687         {
688             for (int j = 0; j < n; j++)
689             {
690                 int l=4;
691                 if (lens_type == 1)
692                     l = 2;
693                 if (lens_type == 2)
694                     l = 6;
695                 if (lens_type == 3){
696                     l = 4;
697                 }
698
699
700                 for (int t = 0; t < l; t++) {
701                     for (int d = 0; d < radial - 1; d++)
702                     {
703                         for (int k = 0; k < tang - 1; k++)
704                         {
705
706                             glBegin(GL_TRIANGLE_STRIP);
707                             x = this->image[i][j].a[k * radial * l + d * l +
    t][0];
708                             y = this->image[i][j].a[k * radial * l + d * l +
    t][1];
709
710                             if (this->image[i][j].a[k * radial * l + d * l +
    t][2] > 0) {
```

```
711                                 COLOR(this->image[i][j].a[k * radial * l + d *
    l + t][3]);
712                                 glColor3f(red * this->image[i][j].a[k * radial
    * l + d * l + t][2], grean * this->image[i][j].a[k * radial * l + d * l +
    t][2], blue * this->image[i][j].a[k * radial * l + d * l + t][2]);
713                                 glVertex3f(x, y, 0);
714                             }
715
716                             x = this->image[i][j].a[k * radial * l + d * l + l
    + t][0];
717                             y = this->image[i][j].a[k * radial * l + d * l + l
    + t][1];
718                             if (this->image[i][j].a[k * radial * l + d * l + l
    + t][2] > 0) {
719                                 COLOR(this->image[i][j].a[k * radial * l + d *
    l + l + t][3]);
720                                 glColor3f(red * this->image[i][j].a[k * radial
    * l + d * l + l + t][2], grean * this->image[i][j].a[k * radial * l + d * l
    + l + t][2], blue * this->image[i][j].a[k * radial * l + d * l + l + t][2]);
721                                 glVertex3f(x, y, 0);
722                             }
723
724
725                             x = this->image[i][j].a[(k + 1) * radial * l + d *
    l + t][0];
726                             y = this->image[i][j].a[(k + 1) * radial * l + d *
    l + t][1];
727                             COLOR(this->image[i][j].a[(k + 1) * radial * l + d
    * l + t][3]);
728                             glColor3f(red * this->image[i][j].a[(k + 1) *
    radial * l + d * l + t][2], grean * this->image[i][j].a[(k + 1) * radial *
    l + d * l + t][2], blue * this->image[i][j].a[(k + 1) * radial * l + d * l
    + t][2]);
729                             if (this->image[i][j].a[(k + 1) * radial * l + d *
    l + t][2] > 0) {
730                                 glVertex3f(x, y, 0);
731                             }
732
733                             x = this->image[i][j].a[(k + 1) * radial * l + d *
    l + t + 1][0];
734                             y = this->image[i][j].a[(k + 1) * radial * l + d *
```

```
l + t + l][1];
735                             COLOR(this->image[i][j].a[(k + 1) * radial * l + d
* l + t + l][3]);
736                             glColor3f(red * this->image[i][j].a[(k + 1) *
radial * l + d * l + t + l][2], grean * this->image[i][j].a[(k + 1) *
radial * l + d * l + t + l][2], blue * this->image[i][j].a[(k + 1) * radial
* l + d * l + t][2]);
737                             if (this->image[i][j].a[(k + 1) * radial * l + d *
l + t + l][2] > 0)
738                             {
739                                 glVertex3f(x, y, 0);
740                             }
741
742                             glEnd();
743                         }
744                     }
745                 }
746             }
747         }
748         if (lens_type == 3) {
749             ray_trasing_for_tor();
750         }
751     }
752     void Show_Images_Sourse_by_point()
753     {
754         glTranslated(-Distanse_im, 0, 0);
755         this->Show_Sourse();
756         glTranslated(2 * Distanse_im, 0, 0);
757         this->Show_Images_by_point();
758         glTranslated(-Distanse_im, 0, 0);
759     }
760     void Show_Images_by_point()
761     {
762         double x, y;
763         for (int i = 0; i < n; i++)
764         {
765             for (int j = 0; j < n; j++)
766             {
767
768                 int l;
769                 if (lens_type == 1)
```

```
                    l = 2;
                if (lens_type == 2)
                    l = 6;
                for (int z = 0; z < NPoints * l; z++) {

                    glBegin(GL_TRIANGLE_STRIP);
                    for (int f = 0; f < 20; f++)
                    {
                        double angle = 2 * 3.13158 / 20 * f;
                        glBegin(GL_TRIANGLE_STRIP);
                        x = this->image[i][j].a[z][0];
                        y = this->image[i][j].a[z][1];

                        if (this->image[i][j].a[z][2] > 0) {
                            glColor3f(1, 0.8, 0);
                            glVertex3f(x + 0.05 / 15* cos(angle), y + 0.05 /
15 * sin(angle), 0);
                        }
                    }
                    glEnd();
                }
            }
        }
    }
    void Show_Images_Sourse() {
        glTranslated(-Distanse_im, 0, 0);
        this->Show_Sourse();
        glTranslated(2 * Distanse_im, 0, 0);
        this->Show_Images();
        glTranslated(-Distanse_im, 0, 0);

    }

    void Show_Images_Rings_Central_Mass() {
        if(this->lens_type==2){
            glColor3f(1, 1, 1);
        glLineWidth(0.1);
        glLineWidth(2);
        glLineWidth(2);
        glBegin(GL_LINE_LOOP);
        for (int i = 0; i < 100; i++) {
```

```
            float theta = 2.0f * 3.14159265 * float(i) / float(100);
            float x1 = r1 * cosf(theta);
            float y1 = r1 * sinf(theta);

            glVertex3f(x1, y1, 0);
        }


        glEnd();

        glBegin(GL_LINE_LOOP);
        for (int i = 0; i < 100; i++) {
            float theta = 2.0f * 3.14159265 * float(i) / float(100);
            float x2 = r2 * cosf(theta);
            float y2 = r2 * sinf(theta);

            glVertex3f(x2, y2, 0);
        }

        glEnd();

        }

        if(this->lens_type==3){
            glColor3f(1, 1, 1);
        glLineWidth(0.1);
        glLineWidth(2);
        glLineWidth(2);
        glBegin(GL_LINE_LOOP);
        for (int i = 0; i < 100; i++) {
            float theta = 2.0f * 3.14159265 * float(i) / float(100);
            float x1 =( Rtor-r0) * cosf(theta);
            float y1 = ( Rtor-r0) * sinf(theta);

            glVertex3f(x1, y1, 0);
        }


        glEnd();

        glBegin(GL_LINE_LOOP);
```

```
851         for (int i = 0; i < 100; i++) {
852             float theta = 2.0f * 3.14159265 * float(i) / float(100);
853             float x2 = ( Rtor+r0) * cosf(theta);
854             float y2 = ( Rtor+r0) * sinf(theta);
855 
856             glVertex3f(x2, y2, 0);
857         }
858 
859         glEnd();
860 
861         }
862 
863     }
864     void Show_Images_and_Sourse_Rings_Central_Mass() {
865         glColor3f(1, 1, 1);
866 
867         glTranslated(-Distanse_im, 0, 0);
868         this->Show_Images_Rings_Central_Mass();
869         this->Show_Central_Mass();
870         glTranslated(2 * Distanse_im, 0, 0);
871         this->Show_Images_Rings_Central_Mass();
872         this->Show_Central_Mass();
873         glTranslated(-Distanse_im, 0, 0);
874 
875     }
876     void Show_Images_and_Sourse_Central_Mass() {
877         glColor3f(1, 1, 1);
878 
879         glTranslated(-Distanse_im, 0, 0);
880         this->Show_Central_Mass();
881         glTranslated(2 * Distanse_im, 0, 0);
882         this->Show_Central_Mass();
883         glTranslated(-Distanse_im, 0, 0);
884     }
885     void Show_Central_Mass() {
886         glColor3f(1, 1, 1);
887         glBegin(GL_TRIANGLE_FAN);
888 
889         for (int h = 0; h < 30; h++)
890         {
891             glVertex3f(0.025 * cos(2 * 3.14159 * h / 30), 0.025 * sin(2 *
```

```
    3.14159 * h / 30), 0);
892         }
893         glEnd();
894     }
895
896     void Show_tor_Central_Mass() {
897         glColor3f(1, 1, 1);
898         glLineWidth(0.1);
899         glLineWidth(2);
900         glLineWidth(2);
901         glBegin(GL_LINE_LOOP);
902         for (int i = 0; i < 100; i++) {
903             float theta = 2.0f * 3.14159265 * float(i) / float(100);
904             float x1 = (Rtor-r0) * cosf(theta);
905             float y1 =( Rtor - r0) * sinf(theta);
906
907             glVertex3f(x1, y1, 0);
908         }
909
910         glEnd();
911         glBegin(GL_LINE_LOOP);
912         for (int i = 0; i < 100; i++) {
913             float theta = 2.0f * 3.14159265 * float(i) / float(100);
914             float x2 = (Rtor + r0) * cosf(theta);
915             float y2 =( Rtor + r0) * sinf(theta);
916
917             glVertex3f(x2, y2, 0);
918         }
919         glEnd();
920     }
921
922     void Show_Marker() {
923         glTranslated(marker, marker, 0);
924         Show_Central_Mass();
925         glTranslated(0, -2 * marker, 0);
926         Show_Central_Mass();
927         glTranslated(-2 * marker, 0, 0);
928         Show_Central_Mass();
929         glTranslated(0, 2 * marker, 0);
930         Show_Central_Mass();
931     }
```

```
932     void Show_Caustic()
933     {
934 
935         glTranslated(-Distanse_im, 0, 0);
936         glLineWidth(10);
937         double r3 = r1 - m0 / r1;
938         double r4 = r2 - 1 / r2;
939         glLineWidth(2);
940 
941         glBegin(GL_LINE_LOOP);
942         for (int i = 0; i < 100; i++) {
943             float theta = 2.0f * 3.14159265 * float(i) / float(100);
944             float x3 = r3 * cosf(theta);
945             float y3 = r3 * sinf(theta);
946             glVertex3f(x3 , y3, 0);
947         }
948 
949         glEnd();
950 
951         glBegin(GL_LINE_LOOP);
952         for (int i = 0; i < 100; i++) {
953             float theta = 2.0f * 3.14159265 * float(i) / float(100);
954             float x4 = r4 * cosf(theta);
955             float y4 = r4 * sinf(theta);
956 
957             glVertex3f(x4, y4, 0);
958         }
959         glEnd();
960     }
961 
962 };
963 
964 int main()
965 {
966 
967     GLFWwindow* window;
968     glfwSetErrorCallback(error_callback);
969     if (!glfwInit())
970         exit(EXIT_FAILURE);
971     window = glfwCreateWindow(1900, 1200, "Simple example", NULL, NULL);
972 
```

```
973
974     if (!window)
975     {
976         glfwTerminate();
977         exit(EXIT_FAILURE);
978     }
979     glfwMakeContextCurrent(window);
980     glfwSetKeyCallback(window, key_callback);
981
982     while (!glfwWindowShouldClose(window))
983     {
984         glFrustum(-1.0, 1, -1, 1, 0, 3);
985         glClear(GL_COLOR_BUFFER_BIT);
986         glMatrixMode(GL_PROJECTION);
987         glLoadIdentity();
988         glMatrixMode(GL_MODELVIEW);
989         glLoadIdentity();
990         glRotatef(0, 1, 0, 0);
991         glEnableClientState(GL_VERTEX_ARRAY);
992
993         glScaled( Scale, - Scale, Scale);
994         glScaled(0.4 / 1900 * 1200, 0.4, 0.4);
995
996         Sourse_Array Grid(3, 1,1);
997         Grid.Show_Images_Sourse();
998         Grid.Show_Images_and_Sourse_Rings_Central_Mass();
999         glTranslated(-Distanse_im, 0, 0);
1000        Grid.Show_Marker();
1001        glTranslated(-Distanse_im, 0, 0);
1002        glTranslated(3*Distanse_im, -Distanse_im, 0);
1003        glfwSwapBuffers(window);
1004        glfwPollEvents();
1005
1006    }
1007    glfwDestroyWindow(window);
1008    glfwTerminate();
1009    exit(EXIT_SUCCESS);
1010    return 0;
1011 }
```

Список використаних джерел

[1] URL: https://science.nasa.gov/mission/hubble.

[2] URL: https://webb.nasa.gov/.

[3] URL: https://www.sdss.org/.

[4] URL: https://www.almaobservatory.org/en/home/.

[5] E. Yu. Bannikova та A. T. Kotvytskiy. "Three Einstein rings: explicit solution and numerical simulation". В: 445.4 (груд. 2014), с. 4438—4445. DOI: 10.1093/mnras/stu2068. arXiv: 1410.7946 [astro-ph.CO].

[6] Albert Einstein. "Lens-like action of a star by the deviation of light in the gravitational field". В: *Science* 84.2188 (1936), с. 506—507.

[7] Raphaël Gavazzi та ін. "The Sloan Lens ACS Survey. VI. Discovery and Analysis of a Double Einstein Ring". В: 677.2 (квіт. 2008), с. 1046—1059. DOI: 10.1086/529541. arXiv: 0801.1555 [astro-ph].

[8] Massimo Meneghetti. *Introduction to gravitational lensing: with Python examples*. Т. 956. Springer Nature, 2021.

[9] Peter Schneider, Jürgen Ehlers та Emilio E. Falco. *Gravitational Lenses*. 1992. DOI: 10.1007/978-3-662-03758-4.