דו"ח מיני פרויקט

מגישים:

יעקב ברין 305416125

אברהם אביחי פרנקל 315390740

תוכן העניינים

מבוא

מטרת המיני-פרויקט:
בשלב אנו כבר יוצרים סצנה תלת מימדית ומדפיסים את התמונה לקובץ.
בפרויקט זה אנו נתנסה בהבנה ותכנות של אובייקטים פשוטים במרחב. אנו נצור מודל המדמה צורות בסיסיות במרחב שלנו.
המימוש בשפת JAVA . ברצוננו להצליח להקרין על המסך תמונה גרפית מציאותית.
בכדי לעשות זאת יש מספר דברים הנדרשים לכך:
• יש לנתח את המציאות הפיזיקאלית ולבחון כיצד היא עובדת.
• יש להתחשב בהרבה גורמים שונים במרחב. לדוגמה: התמונה המשתקפת לאדם באור יום אינה דומה לתמונה בתאורה ביתית. מרחק העצמים שבתמונה ממקור האור. ייצוג של צורות ובניית צורות מורכבות.
• יש צורך לנסות להציג למשתמש תמונה מוחשית ושנראית לו מוכרת, ומצד שני יש להתחשב במגבלות המחשב וחישובים מיותרים שאינם הכרחיים לתחושה שהתמונה המוצגת יפה.
• יש לתכנן כיצד הקוד ייראה וכיצד נוכל להוסיף לו עוד אפשרויות וכלים.
• יש לעשות הכול בזמן קצר כפי שנהוג היום הענף ההיי טק.

כאמור האתגר הוא לתכנן מודל שיענה על כל הדרישות, ויהיה ניתן למימוש.
לשם כך הגדרנו תפקידים מוגדרים לכל חבילה ומחלקה (למשל גיאומטריה יודעת לענו על שאלות כמו מאיזה חומר היא עשויה, מה מתכונות שלו, ובהינתן קרן מהן נקודות החיתוך ועוד הרבה, ומחלקות שמייצגות סוגי אור שונים בצורה שקל לאתר כמה אור מגיע לכל נקודה בזמן הרנדור.)

תחילה נתאר לפי איזה כללים תכנתנו את המיני פרויקט.
א. תכנות בשיטת agile .
ב. תכנות בסיסי ללא שימוש בספריות גרפיקה מוכנות.
ג. הבנה של העולם הפיזיקאלי ומידולו. (אובייקטים, אורות, פעולת המצלמה ע"י עקיבת קרניים)

מבנה כללי:
אלו החבילות העיקריות שמרכיבות את הפרויקט, ע"פ העיקרון התכנותי של התקדמות מהפשוט ביותר עד למורכב:
primitives - מבנים גיאומטרים פשוטים ביותר, שמהווים את התשתית לצורות. בחבילה זו נמצאות המחלקות הבסיסיות לצורך יצירת סצנה גרפית תלת מימדית.
geometries - בחבילה זו הצורות הגיאומטריות הפשוטות, מישור, משולש, כדור.
elements - בחבילה זו נמצאות המחלקות שאחראיות על האלמנטים שגורמים לגופים להיראות כפי שהם נראים בעיני הצופה. זה כולל את המצלמה שהיאו נקודת המבט של הצופה, ואורות שונים, תאורה סביבתית, אור שתלוי במרחק וכדומה.
scene - חבילה זו אחראית לייצג סצנה ספציפית שהמשתמש בוחר ליצור. כאן ישמר אוסף הגופים, התאורות, תאורת הרקע, המצלמה וכדומה.
renderer - באמצעות חבילה זו נעבד את התמונה על כל מרכיביה, ומדפיס אותה לבסוף לקובץ.

חבילות הפרויקט

החבילה primitives:

כוללת את המחלקות הבאות:

Coordinate (קואורדינטה) - יחידה על ציר המספרים:

מכילה נתון _coord מסוג double,
בנאים ופונקציות גישה (שמשתמשים בפונקציות של util לזיהוי ערך שקרוב לאפס כאפס ממש),
פונ' compareTo ((שמחזירה 0 אם שווים. 1 אם גדול ו 1- אם קטן),
פונ' equals (שמחזירה האם שווים וכנ"ל משתמשת בפונקציות של util לזיהוי ערך שקרוב לאפס כאפס ממש),
פונ' subtract - חיסור קואורדינטה מהנוכחית,
פונ' add שמקבלת קואורדינטה ומשנה את הנוכחית כך שתהיה תוצאת החיבור של שתיהן.
פונ' scale שמקבלת מספר ומשנה את הנוכחית כך שתהיה תוצאת הכפל של הנוכחית עם המספר.
פונ' multiply שמקבלת קורואדינטה ומשנה את הנוכחית כך שתהיה תוצאת הכפל שהם.

Point2D (נקודה במישור) - נקודה בעלת 2 קואורדינטות.:

מכילה שתי קואורדינטות,
פונ' compareTo שמשווה נקודות (שמחזירה 0 אם שווים. 1 אם גדול ו 1- אם קטן),
פונ' equals שבודקת האם הנקודות שוות (שימוש בequals של קואורדינטה),
פונ' distance שמקבלת נקודה ומחזירה את המרחק ביניהן,
פונ' distance שלא מקבלת כלום ומחזירה את המרחק של הנקודה מהראשית.

Point3D (נקודה במרחב) - נקודה בעלת 3 קואורדינטות:

יורשת מPoint2D ומוסיפה ערך z (מסוג Coordinate)
פונ' compareTo שמשווה נקודות (שמחזירה 0 אם שווים. 1 אם גדול ו 1- אם קטן),
add - הוספת וקטור .
subtract - חיסור וקטור.
פונ' distance - מקבלת נקודה ומחזירה את המרחק ביניהם.

Vector (וקטור) - ישר שעובר דרך ראשית הצירים ונקודה נתונה במרחב – מגדיר כיוון.

מכיל נקודה _head מסוג Point3D. בנאים: ריק, מקבל Point3D, מקבל Vector, מקבל double 3.
get ו set ל _head .
פונקציות נוספות:
compareTo - ההשוואה בפונקציה זו מתבסס על ההשוואה ב Point3D, לפי אורך הוקטור.
toString - משתמשת ב toString של Point3D.
add - הוספת וקטור לוקטור הנוכחי.
subtract - חיסור וקטור מהוקטור הנוכחי.
scale - הכפלת הוקטור בסקלר.

crossProduct - מחזירה תוצאת מכפלה וקטורית בין 2 וקטורים.
length - אורך הוקטור.
normalize - נירמול הוקטור.
dotProduct - מחזירה תוצאת מכפלה סקלרית בין 2 וקטורים.

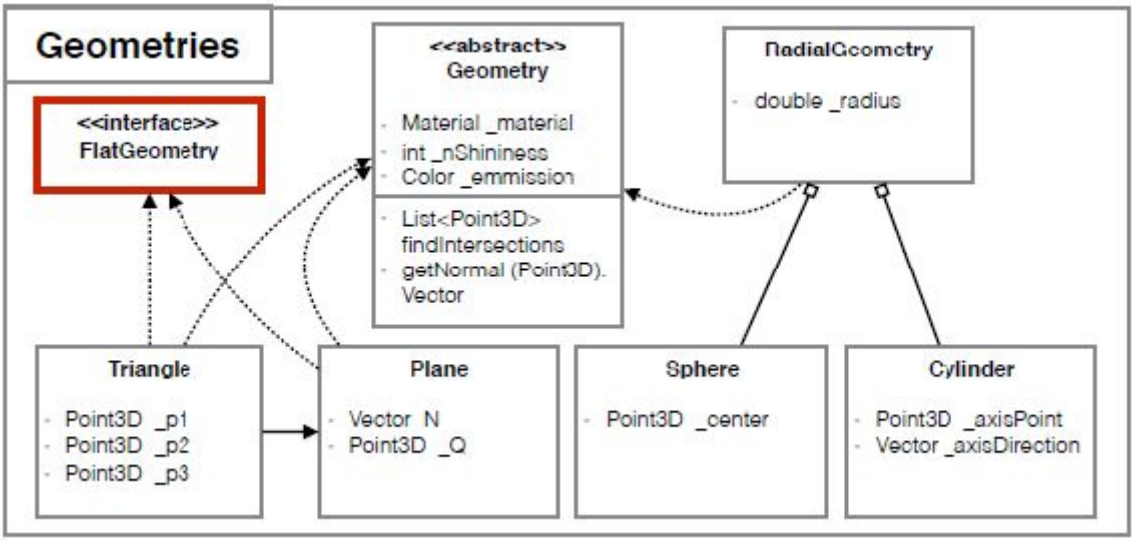
Ray (קרן) - וקטור שאינו עובר בראשית הצירים. מוגדר ע"י נקודה וכיוון (וקטור).

מכיל נקודה _POO מסוג Point3D ו _direction מסוג Vector.
יש בנאי ריק ובנאי העתקה, ו get ו set לכל אחד מהרכיבים.

Material (חומר) - מחלקה זו מכילה תכונות שונות עבור חומר כלשהו, כל תכונה מאופיינת באמצעות פרמטר.

הפרמטרים:
_Kd double - מקדם פיזור.
_Ks double - מקדם צפיפות.
_Kr double - מקדם השתקפות.
_Kt double - מקדם השבירה.
_n double - אינדקס השבירה.

החבילה geometries:



החבילה כוללת את הממשקים הבאים:

Intersectable - מאפשר מציאת נקודות חיתוך של קרן עם אובייקט כלשהו.

מכיל את הפונקציה FindIntersections לביצוע פעולה זו.
הממשק ממומש ע"י מחלקות הגיאומטריות השונות.

FlatGeometry - ממשק למשטחים.

ממשק אופציונלי למשטחים שאינו מכיל פונקציות, ונועד לנוחות המימוש. לא השתמשנו בו בשלב זה.

החבילה כוללת את המחלקות המופשטות הבאות:

Geometry (גיאומטריה) - מאפיינת צורות.

זו מחלקה אבסטרקטית המייצגת משתנים ופעולות של צורה גיאומטרית. למחלקה יש שלושה משתנים, אחד מטיפוס Material המייצג את סוג החומר של הצורה, אחד מטיפוס double המייצג את רמת ההברקה של הצורה ואחד מטיפוס Color המייצג את פליטת הצבע שלה. שלושתם בעלי הרשאת private.
מבנה המחלקה:

```
private Material _material = new Material();
private double _nShininess = 1;
private Color _emmission = new Color(0, 0, 0);
public abstract List<Point3D> FindIntersections (Ray ray);
public abstract Vector getNormal(Point3D point, Vector direction);
public double getShininess();
public Material getMaterial();
public Color getEmmission();
public void setShininess(double n);
public void setMaterial(Material _material);
public void setEmmission(Color emmission);
public void setKs(double ks);
```

```
public void setKd(double kd);
public void setKr(double Kr);
public void setKt(double Kt);
```

findIntersections - פונקציה אבסטרקטית המקבלת כפרמטר משתנה מטיפוס Ray ומחזירה רשימה של נקודות Point3D המייצגים את נקודות החיתוך של הקרן עם הצורה הגיאומטרית.
getNormal - מקבלת כפרמטר משתנה Point3D ווקטור (direction) ומחזירה Vector המייצג את הנורמל, ובכיוון שממנו מגיע הוקטור שקיבלנו. כלומר מאונך לצורה הגיאומטרית בנקודה שהתקבלה כפרמטר.
get/set - מקבלות ומחזירות את הערכים של המשתנים המקומיים ושל פרטי החומר של הצורה.

RadialGeometry (גיאומטריה רדיאלית) - מאפיינת רדיוס.

משמשת לצורות שמאופיינות ברדיוס - כדור וגליל. המחלקה היא אבסטרקטית והיא יורשת מהמחלקה Geometry. למחלקה יש משתנה אחד מטיפוס double בעל הרשאת protected המייצג את אורך הרדיוס של הצורה הגיאומטרית.

מבנה המחלקה:

```
protected double _radius;
public RadialGeometry();
public RadialGeometry(double radius);
public double getRadius();
public void setRadius(double radius);
```

החבילה כוללת את המחלקות הבאות:

Plane (מישור)

המחלקה מייצגת מישור במרחב. המחלקה יורשת מהמחלקה Geometry וממשת את הממשק FlatGeometry. למחלקה יש שני משתנים, אחד מטיפוס Vector המייצג את הנורמל למישור ואחד מטיפוס Point3D המייצג נקודה על המישור. שניהם בעלי הרשאת private.

מבנה המחלקה:

```
private Vector _normal;
private Point3D _Q;
// ***************** Constructors ********************** //
public Plane();
public Plane (Plane plane);
public Plane (Vector normal, Point3D q);
// ***************** Getters/Setters ********************** //
public Vector getNormal(Point3D point, Vector direction);
public Point3D getQ();
public void setNormal(Vector normal);
public void setQ(Point3D d);
// ***************** Operations ******************** //
public List<Point3D> FindIntersections(Ray ray);
```

למחלקה יש שלושה בנאים:

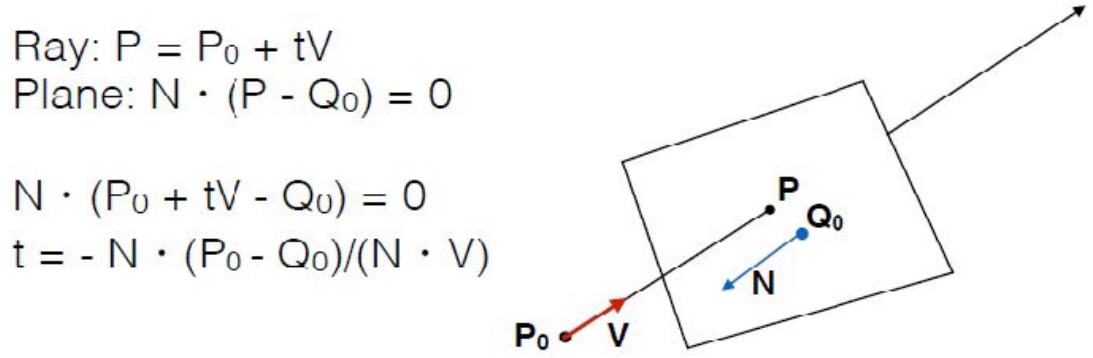
1 בנאי ברירת מחדל שמגדיר את הנורמל בערך (0,0,1) ואת והנקודה בערך (0,0,0)

2 בנאי העתקה.

3 בנאי שמקבל שני ערכים, נורמל ונקודה, ומכניס את ערכם למשתנים המקומיים של המחלקה.

הפונקציות get/set מקבלות ומחזירות את הערכים של המשתנים המקומיים.

הפונקציה findIntersections מקבלת כפרמטר משתנה מטיפוס Ray ומחזירה רשימה בעלת טיפוסים Point3D המייצגים את נקודות החיתוך של הקרן עם המישור. ישנה נקודת חיתוך אחת או שאין בכלל. צורת החישוב:



Sphere (כדור)

המחלקה מייצגת כדור במרחב. המחלקה יורשת ממחלקת RadialGeometry . למחלקה יש משתנה אחד מטיפוס Point3D בעל הרשאת private המייצג את מרכז הכדור.

מבנה המחלקה:

```
private Point3D _center;
// ***************** Constructors ********************** //
public Sphere();
public Sphere (Sphere sphere);
public Sphere(double radius, Point3D center);
public Sphere(Map<String, String> attributes);
// ***************** Getters/Setters ********************** //
public Point3D getCenter();
public void setCenter(Point3D center);
// ***************** Operations ******************** //
public List<Point3D> FindIntersections(Ray ray);
public Vector getNormal(Point3D point, Vector direction);
```

למחלקה יש שלושה בנאים:

1 בנאי ברירת מחדל שמגדיר את הרדיוס באורך 0 ואת הנקודה בערך (0,0,0).

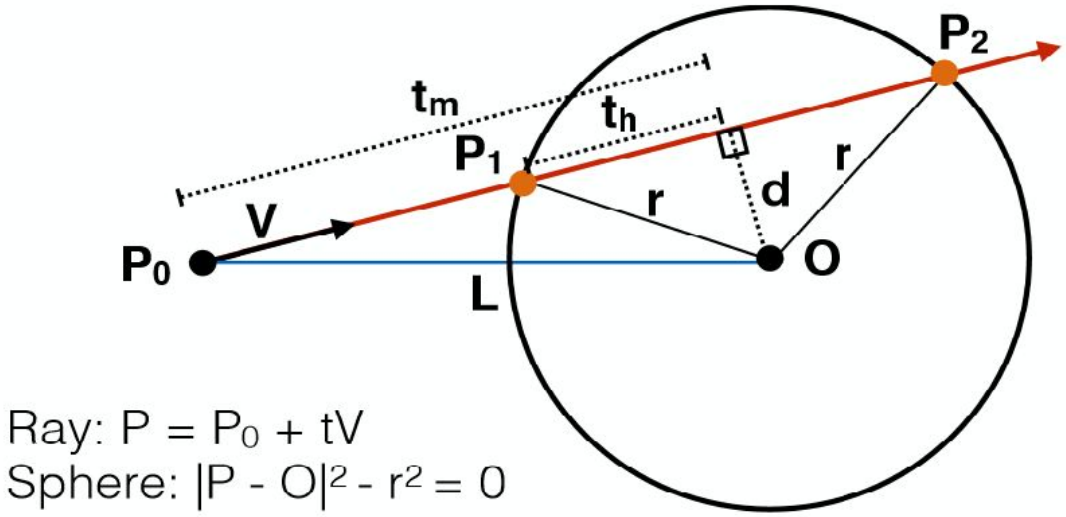
2 בנאי העתקה.

3 בנאי שמקבל שני ערכים, רדיוס מטיפוס double ונקודת מרכז מטיפוס Point3D, ומכניס את ערכם למשתנים המקומיים של המחלקה.

get/set מקבלות ומחזירות את הערך של נקודת המרכז המקומית.

getNormal מקבלת כפרמטר משתנה מטיפוס Point3D ווקטור (direction) ומחזירה ערך מטיפוס Vector המייצג את הנורמל לכדור בנקודה שהתקבלה כפרמטר ובכיוון שממנו מגיע הוקטור direction.

findIntersections מקבלת כפרמטר משתנה מטיפוס Ray ומחזירה רשימה בעלת טיפוסים Point3D המייצגים את נקודות החיתוך של הקרן עם הכדור. ישנן שתי נקודות חיתוך, נקודת חיתוך אחת או שאין בכלל.



הסבר קצר על האיור - P0 זוהי נקודת מיקום המצלמה.
O – אמצע הכדור, r – רדיוס הכדור, P1+P2 – נקודות החיתוך של הכדור עם ה-ray.
תהליך חישוב נקודות החיתוך:

1. מציאת הוקטור L: L = O - P0
2. מציאת tm דהיינו ההיטל של L על הקרן: tm = L . V)מכפלה סקלרית(.
3. מציאת d – המרחק בין tm למרכז המעגל: d = (|L|^2 - tm^2)^0.5
4. בדיקה האורך d: אם המרחק גדול מהרדיוס אזי אין כלל נקודת חיתוך ותוחזר רשימה ריקה.
   אם המרחק שווה לרדיוס הפונקציה תחזיר נקודת חיתוך אחת.
   אם המרחק בין 0 לרדיוס ישנם שתי נקודות חיתוך. הפונקציה תחזיר אותם.
5. חישוב th (שזה חצי מהקשת שחותכת את העיגול) : th = (r^2 - d^2)^0.5
6. חישוב של t1,t2: המרחק של הקרן המנורמלת מנקודת החיתוך.
7. שליחה לפונקציית העזר בכדי לחשב את נקודת החיתוך.

getPoint פונקציה פנימית של המחלקה (private) ונועדה לחשב את נקודת החיתוך עבור הכדור. הפונקציה נצרכת משום שפעמים יש שתי נקודות חיתוך וע"י אותו חישוב (עם פרמטרים שונים) ניתן למצוא אותם.
הפונקציה מקבלת שלושה פרמטרים: t - המרחק בין הקרן המנורמלת לנקודת החיתוך. v- הכיוון של הקרן (מנורמל), poo – נקודת מיקום המצלמה. הפונקציה מחשבת את נקודת החיתוך ע"י הנוסחה הבאה:
P0 (P = P0 + tV הכוונה ל-poo), ומחזירה את הנקודה שנמצאה.

Hemisphere (חצי כדור)

המחלקה מייצגת קערה - חצי כדור במרחב. המחלקה יורשת ממחלקת RadialGeometry . למחלקה יש שני משתנים מטיפוס Point3D בעלי הרשאת private, אחד המייצג את מרכז הקערה, ואחד המייצג את תחתית הקערה.
מבנה המחלקה:

```
private Point3D _center;
private Point3D _bottom;
// ***************** Constructors ********************** //
```

```
public Hemisphere();
public Hemisphere(Hemisphere hemisphere);
public Hemisphere(double _radius, Point3D _center, Vector _vBottom);
// ***************** Getters/Setters ********************** //
public void set_bottom(Point3D _bottom);
 public Point3D getBottom();
public Point3D getCenter();
public Hemisphere set_center(Point3D _center);
// ***************** Operations ******************** //
public List<Point3D> FindIntersections(Ray ray);
public Vector getNormal(Point3D point, Vector direction);
```

למחלקה יש שלושה בנאים:

1. בנאי ברירת מחדל שמגדיר את הרדיוס באורך 0 ואת שתי הנקודות בערך (0,0,0).
2. בנאי העתקה.
3. בנאי שמקבל שלושה ערכים, רדיוס מטיפוס double, נקודת מרכז מטיפוס Point3D ווקטור לכיוון החלק הפתוח של הקערה. מחשב את נקודת התחתית של הקערה, ומכניס את ערכים למשתנים המקומיים של המחלקה.

get/set מקבלות ומחזירות את הערך של נקודת המרכז או התחתית המקומית.

getNormal מקבלת כפרמטר משתנה מטיפוס Point3D ווקטור (direction) ומחזירה ערך מטיפוס Vector המייצג את הנורמל לקערה בנקודה שהתקבלה כפרמטר ובכיוון שממנו מגיע הוקטור direction.

findIntersections מקבלת כפרמטר משתנה מטיפוס Ray ומחזירה רשימה בעלת טיפוסים Point3D המייצגים את נקודות החיתוך של הקרן עם הקערה. ישנן שתי נקודות חיתוך, נקודת חיתוך אחת או שאין בכלל.

Triangle (משולש)

המחלקה מייצגת משולש במרחב. המחלקה יורשת מהמחלקה Geometry וממשת את הממשק FlatGeometry . למחלקה יש שלושה משתנים מטיפוס Point3D המייצגים את שלושת הנקודות של המשולש. שלושתם בעלי הרשאת private.

מבנה המחלקה:

```
private Point3D _p1;
private Point3D _p2;
private Point3D _p3;
// ***************** Constructors ********************** //
public Triangle();
public Triangle(Triangle triangle);
public Triangle(Point3D p1, Point3D p2, Point3D p3);
public Triangle(Map<String, String> attributes);
// ***************** Getters/Setters ********************** //
public Point3D getP1();
public Point3D getP2();
public Point3D getP3();
public void setP1(Point3D p1);
public void setP2(Point3D p2);
public void setP3(Point3D p3);
// ***************** Operations ******************** //
```

```
public Vector getNormal(Point3D point, Vector direction);
public List<Point3D> FindIntersections(Ray ray);
```

למחלקה יש שלושה בנאים:

1 בנאי ברירת מחדל שמגדיר את הנקודות בערכים (0,0,0) ,(1,0,0) ו-(0,0,1).

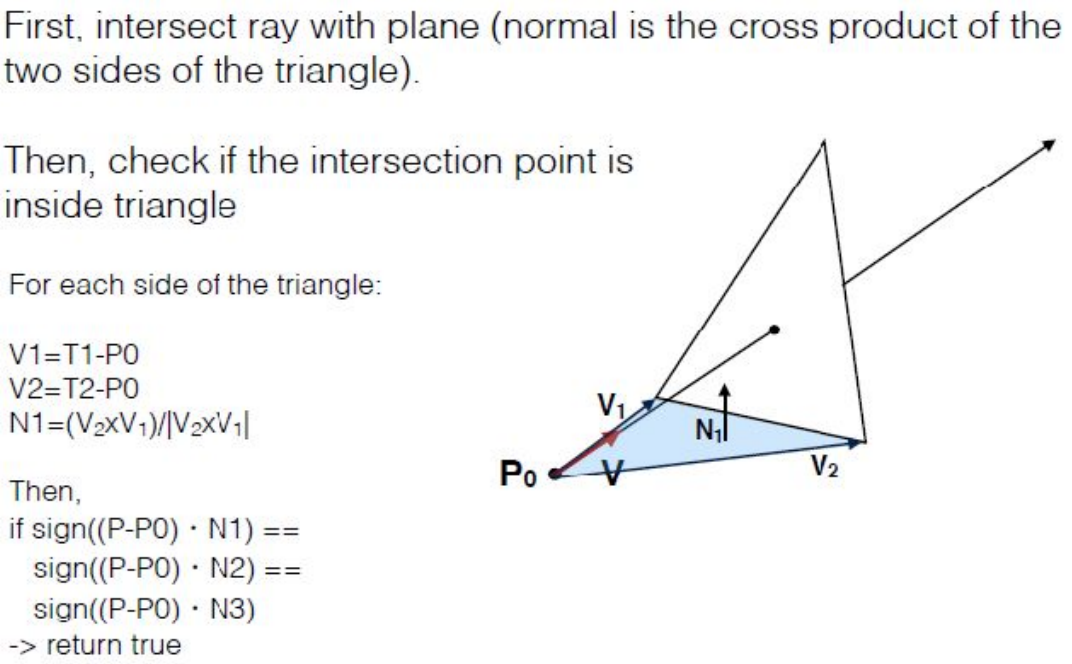
2 בנאי העתקה.

3 בנאי שמקבל שלושה ערכים מטיפוס Point3D ומכניס את ערכם לנקודות של המשתנה המקומי.

get/set מקבלות ומחזירות את הערכים של המשתנים המקומיים.

getNormal מקבלת כפרמטר משתנה מטיפוס Point3D (במקרה הזה הוא לא רלוונטי) ווקטור (direction) ומחזירה Vector המייצג את הנורמל למשולש ובכיוון שממנו מגיע הוקטור direction.

findIntersections מקבלת Ray ומחזירה רשימה של Point3D המייצגים את נקודות החיתוך של הקרן עם המשולש. ישנה נקודת חיתוך אחת או שאין בכלל. מחשבים זאת ע"י שיוצרים מהמשולש מישור, ובודקים ע"י שלושה נורמלים האם נקודת החיתוך שבמישור נמצאת גם במשולש.



getSign היא פונקצית עזר (private) שמקבלת כפרמטר נקודת מקור מטיפוס Point3D , שתי נקודות של המשולש מטיפוס Point3D ווקטור מטיפוס Vector בין נקודת המקור לנקודת החיתוך. הפונקציה מחזירה ערך מסוג double המייצג ערך המכפלה הסקלרית בין הנורמל לשתי הנקודות של המשולש, לבין הוקטור בין נקודת המקור לנקודת החיתוך. אם בשלוש הפעמים של הפעלת הפונקציה יוצא מספר חיובי או שלילי, אזי נקודת החיתוך נמצאת בתוך המשולש.

Quadrangle (מרובע)

המחלקה מייצגת מרובע במרחב. המחלקה יורשת מהמחלקה Geometry וממשת את הממשק FlatGeometry. המרובע מיוצג ע"י שני משולשים. למחלקה יש שני משתנים מטיפוס Triangle המייצגים את שלושת הנקודות של המשולש. שלושתם בעלי הרשאת private.

מבנה המחלקה:

```
private Triangle _tri1;
```

```
private Triangle _tri2;
public Quadrangle(Point3D p1, Point3D p2, Point3D p3, Point3D p4);
public Quadrangle(Quadrangle quadrangle);
public Triangle get_tri1();
public Vector getNormal(Point3D point_NoUse);
public List<Point3D> FindIntersections(Ray ray) ;
```

למחלקה יש שני בנאים:

1 בנאי המקבל ארבע נקודות ומאתחל את קודקודי המרובע.

2 בנאי העתקה.

get/set מקבלות ומחזירות את הערכים של המשתנים המקומיים. (כמובן שאין אפשרות לשנות את אחד המשולשים)

getNormal מקבלת כפרמטר משתנה מטיפוס Point3D (במקרה הזה הוא לא רלוונטי) ווקטור (direction) ומחזירה Vector המייצג את הנורמל למרובע ובכיוון שממנו מגיע הוקטור direction.משתמשת בפונקציה getNormal של Triangle.

findIntersections מקבלת Ray ומחזירה רשימה של Point3D המייצגים את נקודות החיתוך של הקרן עם המרובע. ישנה נקודת חיתוך אחת או שאין בכלל. משתמשת בפונקציה findIntersections של Triangle.

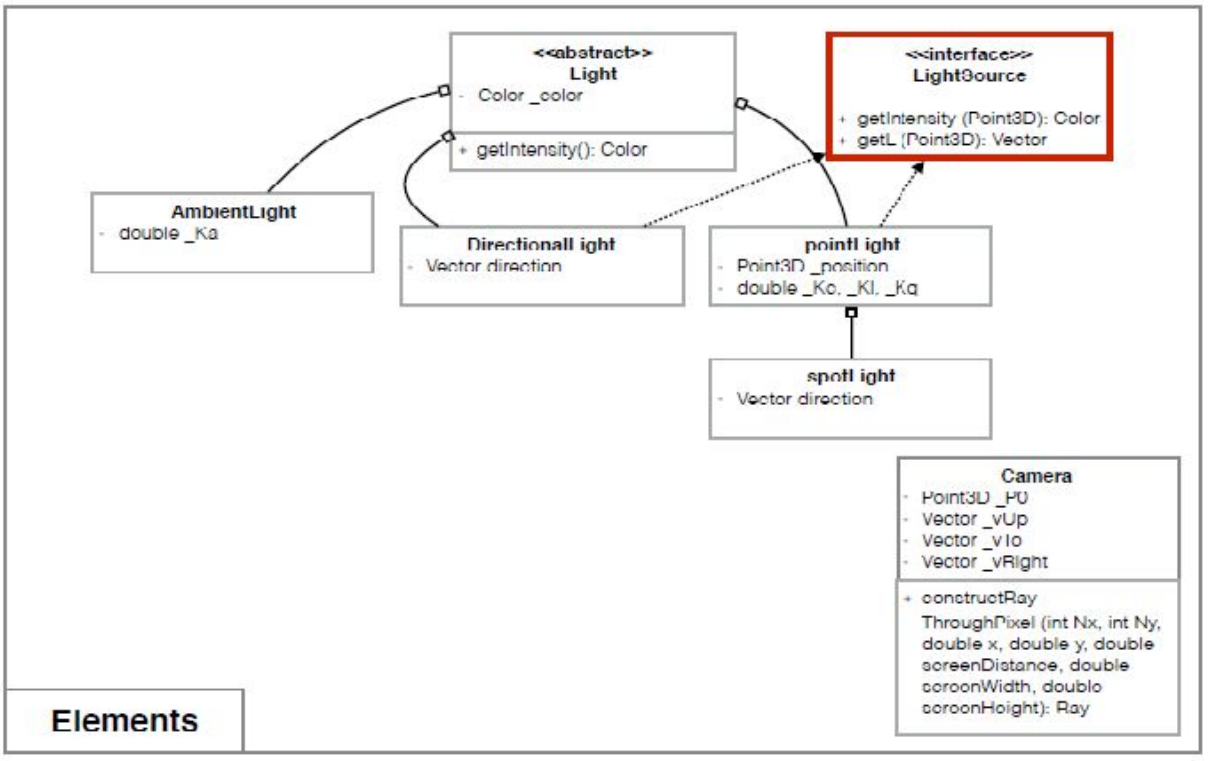
Cylinder (גליל)

במחלקה זו ישנו בסיס למימוש עתידי של גליל, אבל אינו שימושי בשלב זה, בשל מורכבות המימוש שלו.
מבנה המחלקה:

```
private Point3D _axisPoint;
private Vector _axisDirection;
// ***************** Constructors ********************** //
public Cylinder();
public Cylinder(Cylinder cylinder);
public Cylinder(double radius, Point3D axisPoint, Vector axisDirection);
// ***************** Getters/Setters ********************** //
public Point3D getAxisPoint();
public Vector getAxisDirection();
public void setAxisPoint(Point3D axisPoint);
public void setAxisDirection(Vector axisDirection);
// ***************** Operations ******************** //
public List<Point3D> FindIntersections(Ray ray);
public Vector getNormal(Point3D point, Vector direction);
```

החבילה elements:

חבילה המייצגת את המצלמה של הסצנה ואת מקורות האור בהם אנו משתמשים בסצנה.



כוללת את המחלקות והממשקים הבאים:

Camera (מצלמה) -

מבנה המחלקה:

```
//Eye point of the camera
private Point3D _P0;
private Vector _vUp;
private Vector _vTo;
//Should be calculated as the cross product if vUp and vTo
private Vector _vRight;
// ***************** Constructors ********************** //
public Camera(); ;
public Camera (Camera camera(;
public Camera (Point3D P0, Vector vUp, Vector vTo(;
public Camera (Map<String, String> attributes(;
// ***************** Getters/Setters ********************** //
public Vector get_vUp)(;
public void set_vUp(Vector vUp(;
public Vector get_vTo)(;
public void set_vTo(Vector vTo(;
public Point3D getP0)(;
public void setP0(Point3D P0);
public Vector get_vRight)(;
```

```
// ***************** Administration ********************** //
public String toString)(;
// ***************** Operations ******************** //
public Ray constructRayThroughPixel (int Nx, int Ny,
double x, double y,
double screenDist,
double screenWidth,
double screenHeight);
```

למחלקה יש שלושה בנאים:

1 בנאי ברירת מחדל שמגדיר את עין המצלמה בנקודה (0,0,10), את הכיוון מעלה בערך (0,1,0), את הכיוון מול בערך (0,0,-1), ואת הכיוון ימינה בערך המכפלה הסקלרית בין הכיוון מעלה והכיוון מול.

2 בנאי העתקה.

3 בנאי שמקבל שלושה ערכים, אחד מטיפוס Point3D ושניים מטיפוס Vector ומכניס את ערכם למשתנים המקומיים של המחלקה.

get/set מקבלות ומחזירות את הערכים של המשתנים המקומיים.

toString מדפיסה את נתוני המצלמה.

constructRayThroughPixel מקבלת כפרמטרים שבעה משתנים, שניים מטיפוס int המייצגים את מספר הפיקסלים במסך, וחמישה מטיפוסdouble המייצגים את אורכי המסך ואת הפיקסל המבוקש. היא מחזירה ערך מסוג Ray המייצג את הקרן המתחילה מעין המצלמה ומגיעה לנקודה שבה הפיקסל המבוקש.

הנוסחה מחושבת בצורה הבאה:

Image center

$P_c = P_0 + dV_{to}$

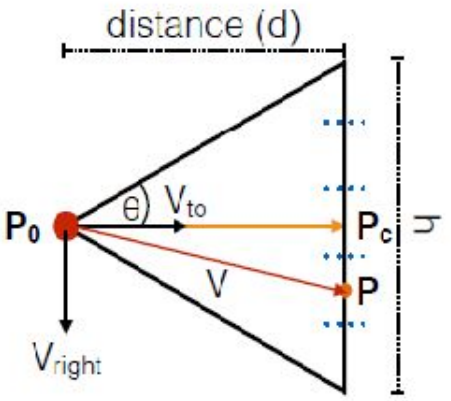
Vright

$V_{right} = V_{to} \times V_{up}$

Ratio (pixel width)

$R_x = w/\#Pixels_x$

$R_y = h/\#Pixels_y$



$$P = P_C + \left[\left[\left(x - \frac{\#\,pixels_x}{2}\right)R_x + \frac{R_x}{2}\right]V_{right} - \left[\left(y - \frac{\#\,pixels_y}{2}\right)R_y + \frac{Ry}{2}\right]V_{up}\right]$$

Light

המחלקה מייצגת אור. המחלקה היא אבסטרקטית. למחלקה יש משתנה אחד מטיפוס Color בעל הרשאת protected המייצג את צבע האור.

מבנה המחלקה:

```
protected Color _color;
// ***************** Constructors ********************** //
public Light();
public Light (Color color);
// ***************** Getters/Setters ********************** //
public Color getIntensity();
```

למחלקה יש שני בנאים:

1. בנאי ברירת מחדל שמגדיר את הצבע כצבע שחור.
2. בנאי שמקבל ערך מטיפוס Color ומכניס את ערכו למשתנה הצבע של המחלקה.

הפונקציה **getIntensity** מחזירה ערך מטיפוס Color המייצג את עוצמת האור ע"פ צבעו.
הפונקציות **get/set** מקבלות ומחזירות את הערך של הצבע המקומי.

LightSource (ממשק)

מייצג מקורות אור חיצוניים. מבנה הממשק:

```
public abstract Color getIntensity(Point3D point);
public abstract Vector getL(Point3D point);
```

הפונקציה **getIntensity** מקבלת כפרמטר משתנה מטיפוס Point3D ומחזירה ערך מטיפוס Color המייצג את עוצמת האור בנקודה זו.
הפונקציה getL מקבלת כפרמטר משתנה מסוג Point3D ומחזירה ערך מטיפוס Vector המייצג את הוקטור ממקור האור לנקודה זו.

AmbientLight (תאורה סביבתית) -

המחלקה מייצגת אור סביבתי. המחלקה יורשת מהמחלקה Light. למחלקה יש צבע ועוד משתנה אחד מטיפוס double בעל הרשאת private המייצג קבוע המחליש את האור.
מבנה המחלקה:

```
private final double _Ka = 0.1;
// ***************** Constructors ********************** //
public AmbientLight();
public AmbientLight(AmbientLight aLight);
public AmbientLight(int r, int g, int b);
public AmbientLight(Map<String, String> attributes);
// ***************** Getters/Setters ********************** //
public Color getColor();
public void setColor(Color color);
public double getKa();
public Color getIntensity();
```

למחלקה יש ארבעה בנאים:

1 בנאי ברירת מחדל שמגדיר את הצבע כצבע שחור.

2 בנאי העתקה.
3 בנאי המקבל שלושה ערכים מטיפוס int המייצגים את עוצמת הצבעים אדום, ירוק וכחול, ובונה מהם את צבע המשתנה של המחלקה.
4 בנאי המקבל שלושה ערכים מטיפוס double המייצגים את עוצמת הצבעים אדום, ירוק וכחול, ובונה מהם את צבע המשתנה של המחלקה.

get/set מקבלות ומחזירות את הערכים של המשתנים המקומיים.

getIntensity מחזירה ערך מסוג Color המייצג את עוצמת האור הסביבתי.

DirectonalLight

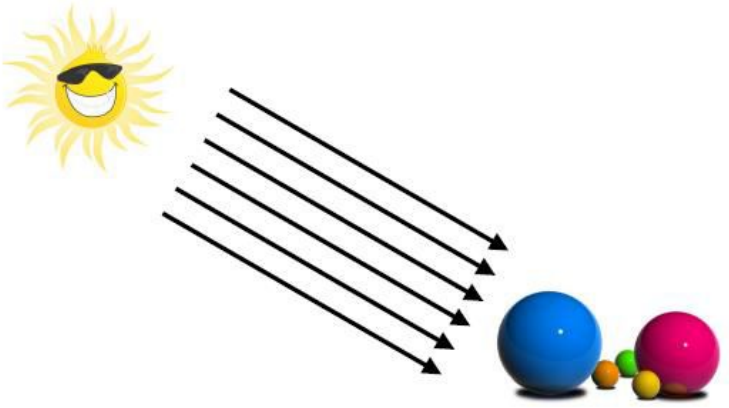
המחלקה מייצגת אור עם כיוון וללא חשיבות למיקום (מיקום באינסוף, כמו שמש). המחלקה יורשת מהמחלקה Light וממשת את הממשק LightSource .

למחלקה יש משתנה אחד מטיפוס Vector בעל הרשאת private , המייצג את כיוון האור. מבנה המחלקה:

```
private Vector _direction;
// ***************** Constructors ********************** //
public DirectionalLight(Color color, Vector direction);
// ***************** Getters/Setters ********************** //
public Color getIntensity(Point3D point);
public Vector getDirection();
public void setDirection(Vector _direction);
public Vector getL(Point3D point);
```

למחלקה יש בנאי אחד המקבל שני ערכים, צבע מטיפוס Color וכיוון מטיפוס Vector ומכניס את ערכם למשתנים המקומיים של המחלקה.

הפונקציות get/set מקבלות ומחזירות את הערכים של המשתנים המקומיים.



PointLight

המחלקה מייצגת נקודה המפיקה אור (כמו נורה, ללא חשיבות לכיוון). המחלקה יורשת מהמחלקה Light וממשת את הממשק LightSource . למחלקה יש ארבעה משתנים, אחד מטיפוס Point3D המייצג את מיקום האור, ושלושה מטיפוס double המייצגים את גורמי הנחתת האור. כולם בעלי הרשאת protected . מבנה המחלקה:

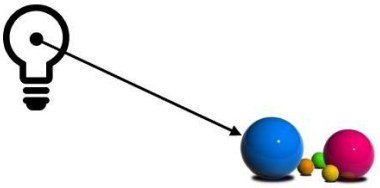
```
private Point3D _position;
```

```
private double _Kc, _Kl, _Kq;
// ***************** Constructors ********************** //
public PointLight(Color color, Point3D position,
double kc, double kl, double kq);
// ***************** Getters/Setters ********************** //
public Color getIntensity(Point3D point);
public Vector getL(Point3D point);
```

למחלקה יש בנאי אחד המקבל חמישה ערכים, צבע מטיפוס Color , מיקום מטיפוס Point3D ושלושה גורמי הנחתה מטיפוס double , ומכניס את ערכם למשתנים המקומיים של המחלקה.
הפונקציה getIntensity מקבלת כפרמטר משתנה מטיפוס Point3D ומחזירה ערך מטיפוס Color המייצג את עוצמת האור בנקודה זו. החישוב מבוצע באמצעות הנוסחה:

$$I_L = I_0 / (K_c \cdot k_j d \cdot k_q d^2)$$

הפונקציה getL מקבלת כפרמטר משתנה מסוג Point3D ומחזירה ערך מטיפוס Vector המייצג את הוקטור מנקודת האור לנקודה זו.



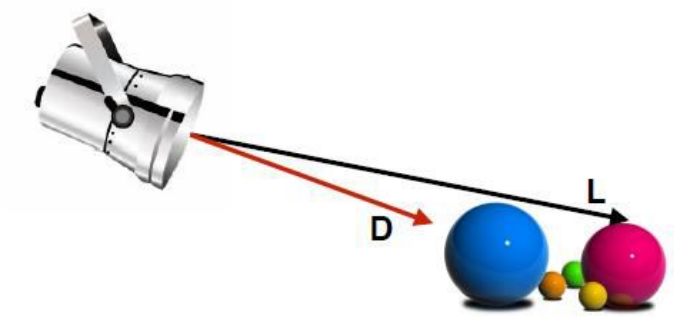
SpotLight

המחלקה מייצגת נקודה המפיקה אור עם כיוון (כמו פרוז'קטור). המחלקה יורשת מהמחלקה PointLight .
למחלקה יש משתנה אחד מטיפוס Vector בעל הרשאת private , המייצג את כיוון האור. מבנה המחלקה:

```
private Vector _direction;
// ***************** Constructor ********************** //
public SpotLight(Color color, Point3D position, Vector direction,
double kc, double kl, double kq);
// ***************** Getters/Setters ********************** //
public Color getIntensity(Point3D point);
```

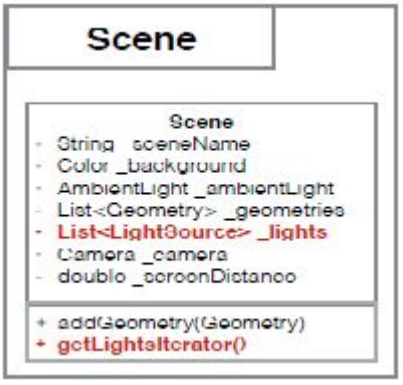
למחלקה יש בנאי אחד המקבל שישה ערכים, צבע מטיפוס Color , מיקום מטיפוס Point3D , כיוון מטיפוס Vector ושלושה גורמי הנחתה מטיפוס double , ומכניס את ערכם למשתנים המקומיים של המחלקה.
הפונקציה getIntensity מקבלת כפרמטר משתנה מטיפוס Point3D ומחזירה ערך מטיפוס Color המייצג את עוצמת האור בנקודה זו. החישוב מבוצע באמצעות הנוסחה:

$$I_L = [I_0 \underbrace{\boxed{(D \cdot L)}}_{\text{dot product}}] / (K_c \cdot k_j d \cdot k_q d^2)$$


L
D

החבילה scene:

חבילה המייצגת את מכלול הסצנה ע"י ריכוז הצורות הגיאומטריות בתמונה, התאורה, הרקע וכו'...
מבנה החבילה:



כוללת את המחלקות הבאות:

Scene (סצנה) -

המחלקה מייצגת סצנה. למחלקה יש שבעה משתנים, אחד מטיפוס Color המייצג את צבע הרקע של הסצנה, אחד מטיפוס AmbientLight המייצג את התאורה הסביבתית של הסצנה, אחד מטיפוס List<Geometry> המייצג רשימה של הצורות הגיאומטריות שבסצנה, אחד מטיפוס Camera המייצג את המצלמה שמצלמת את הסצנה, אחד מטיפוס double המייצג את המרחק בין המצלמה למסך, אחד מטיפוס List<LightSource> המייצג רשימה של מקורות האור של הסצנה ואחד מטיפוס String המייצג את שם הסצנה. כולם בהרשאת private.
מבנה המחלקה:

```
private Color _background;
private AmbientLight _ambientLight;
private List<Geometry> _geometries = new ArrayList<Geometry>();
private Camera _camera;
private double _screenDistance;
private List<LightSource> _lights = new ArrayList<LightSource>();
private String _sceneName = "scene";
// ***************** Constructors ********************** //
public Scene();
public Scene (Scene scene);
public Scene(AmbientLight aLight, Color background,Camera camera, double
screenDistance);
// ***************** Getters/Setters ********************** //
public Color getBackground();
public AmbientLight getAmbientLight();
public Camera getCamera();
public String getSceneName();
public double getScreenDistance();
public void setBackground(Color _background);
public void setAmbientLight(AmbientLight ambientLight);
public void setCamera(Camera camera);
```

```
public void setSceneName(String sceneNAme);
public void setScreenDistance(double screenDistance);
// ***************** Operations ******************** //
public void addGeometry(Geometry geometry);
public Iterator<Geometry> getGeometriesIterator();
public void addLight(LightSource light);
public Iterator<LightSource> getLightsIterator();
```

למחלקה יש שלושה בנאים:

1 בנאי ברירת מחדל שמגדיר את כל המשתנים כברירת המחדל שלהם, ואת מרחק המסך בתור 100.

2 בנאי העתקה.

3 בנאי המקבל ארבעה ערכים, אור סביבתי מטיפוס AmbientLight , צבע רקע מטיפוס Color , מצלמה מטיפוס Camera ומרחק בין המצלמה למסך מטיפוס double, ומכניס את ערכם למשתנים המקומיים של המחלקה.

get/set מקבלות ומחזירות את הערכים של המשתנים המקומיים.

addGeometry מקבלת כפרמטר משתנה מטיפוס Geometry ומכניסה אותו לרשימת הצורות הגיאומטריות של הסצנה.

getGeometriesIterator מחזירה איטרטור לרשימת הצורות הגיאומטריות של הסצנה.

החבילה renderer:

חבילה המייצגת את מחולל התמונה, המחשבת את הצבע המתאים לכל ומייצרת תמונה מתאימה.
החבילה כוללת את המחלקות הבאות:

ImageWriter (כותב התמונה) -

המחלקה מייצגת את המדפיס של התמונה. למחלקה יש שבעה משתנים, ארבעה מטיפוס int המייצגים גודל התמונה ומספר הפיקסלים שבה, קבוע מטיפוס String המייצג את מיקום הקובץ במחשב, אחד מטיפוס BufferedImage המייצג את התמונה עצמה ואחד מטיפוס String המייצג את שם התמונה. כולם בהרשאת private.

מבנה המחלקה:

```
private int _imageWidth;
private int _imageHeight;
private int _Ny, _Nx;
final String PROJECT_PATH = System.getProperty("user.dir");
private BufferedImage _image;
private String _imageName;
// ***************** Constructors ********************** //
public ImageWriter(String imageName, int width, int height,
int Ny, int Nx);
public ImageWriter(ImageWriter imageWriter);
// ***************** Getters/Setters ********************** //
public int getWidth();
public int getHeight();
public int getNy();
public int getNx();
public void setNy(int _Ny);
public void setNx(int _Nx);
// ***************** Operations ******************** //
public void writeToimage();
public void writePixel(int xIndex, int yIndex, int r, int g, int b);
public void writePixel(int xIndex, int yIndex, int[] rgbArray);
public void writePixel(int xIndex, int yIndex, Color color);
```

למחלקה יש שני בנאים:

1 בנאי המקבל חמישה ערכים, ארבעה המייצגים את גודל הסמך ומספר הפיקסלים שבו מטיפוס int ואחד מטיפוס String המייצג את שם התמונה, ומכניס את ערכם למשתנים המקומיים של המחלקה.

2 בנאי העתקה.

get/set מקבלות ומחזירות את הערכים של המשתנים המקומיים.

writeToImage מייצרת את התמונה לתוך קובץ מסוג jpg.

writeToPixel מקבלת כפרמטר שני משתנים מטיפוס int המייצגים את הפיקסל ומשתנה המייצג צבע (בפונקציה אחת הוא מיוצג באמצעות שלושה משתנים מטיפוס int , בשנייה באמצעות מערך מטיפוס int ובשלישית באמצעות משתנה מטיפוס Color), ומציירת בפיקסל הנתון את הצבע שהיא קיבלה.

Render (מעבד) -

זוהי מחלקת הליבה של הפרויקט. מחלקה זו מייצרת לנו את התמונה ע"י קבלת הנתונים ממחלקת ה-Scene ועיבודם עד הצגת התמונה בשלמותה. הצגת התמונה כוללת פרטים רבים מאוד כגון הצורות הגיאומטריות הקיימות בסצנה, המרחקים מכל צורה וצורה, מציאת המרחק הקטן ביותר, הצבע לכל צורה, האורות המשפיעים ביחס לצבע ועוד. למחלקה יש שלושה משתנים, אחד מטיפוס Scene המייצג את הסצנה של התמונה, אחד מטיפוס ImageWriter המייצג את המדפיס של התמונה וקבוע מטיפוס int המייצג את רמת הרקורסיה. כולם בהרשאת private.

מבנה המחלקה:

```
private Scene _scene;
private ImageWriter _imageWriter;
private final int RECURSION_LEVEL = 3;
// ***************** Constructors ********************** //
public Render(ImageWriter imageWriter, Scene scene);
// ***************** Operations ******************** //
public void renderImage();
private Entry<Geometry, Point3D> findClosesntIntersection(Ray ray);
public void printGrid(int interval);
public void writeToImage();
private Color calcColor(Geometry geometry, Point3D point, Ray ray);
private Color calcColor(Geometry geometry, Point3D point,
Ray inRay, int level); // Recursive
private Ray constructRefractedRay(Geometry geometry, Point3D point,
Ray inRay);
private Ray constructReflectedRay(Vector normal, Point3D point,
Ray inRay);
private boolean occluded(LightSource light, Point3D point,
Geometry geometry);
private Color calcSpecularComp(double ks, Vector v, Vector normal,
Vector l, double shininess, Color lightIntensity);
private Color calcDiffusiveComp(double kd, Vector normal, Vector l,
Color lightIntensity);
private Map<Geometry, Point3D> getClosestPoint(Map<Geometry,
List<Point3D>> intersectionPoints);
private Map<Geometry, List<Point3D>> getSceneRayIntersections(Ray ray);
private Color addColors(Color a, Color b);
```

למחלקה יש בנאי אחד המקבל שני ערכים: ImageWriter ו- Scene, ומכניס את ערכם למשתנים המקומיים של המחלקה.

הפונקציה **renderImage** מקבלת את נקודות החיתוך ומוצאת את הצבע המתאים לכל פיקסל. לכל נקודה על המסך מבצעים את האלגוריתם הבא:
שולחים קרניים מהמצלמה לכל נקודה על המסך. את הקרן שומרים במשתנה מטיפוס Ray.
כדי למצוא את הקרן משתמשים בפונקצית constructRayThroughPixel השייכת למחלקת המצלמה.
קוראים לפונקציה getSceneRayIntersections שמחזירה את נקודות החיתוך עם הקרן, אם קיימות.
אם אין נקודות חיתוך, הכנס למשתנה imageWriter את הנקודה הנתונה ואת צבע הרקע.
אם יש נקודת חיתוך אזי:
מצא את הצבע שצריך להיות בנקודה הקרובה ע"י הפונקציה calcColor. אנו שולחים כפרמטרים את ה-Geometry ואת ה-Point3D שהתקבל בנקודת החיתוך הקרובה ביותר.
הכנס את הנקודה וצבעה למשתנה imageWriter.

הפונקציה **findClosestIntersection** מקבלת כפרמטר משתנה מטיפוס Ray המייצג את הקרן היוצאת מהמצלמה, ומחזירה ערך מטיפוס <Entry<Geometry, Point3D המייצג את המיקום במפה של נקודת החיתוך. הקרובה ביותר. הפונקציה קוראת לפונקציה getSceneRayIntersections ובעזרתה היא מייצרת Map , השומרת לנו ב- key שלה את הצורות הגיאומטריות שנחתכות עם הקרן וב- value שלה רשימה המונה את נקודות החיתוך שלהן. אם אין נקודות חיתוך היא מחזירה null , אחרת, היא קוראת לפונקציה getClosestPoint ומחזירה את הנקודה הקרובה ביותר.

הפונקציה **getSceneRayIntersections** מקבלת כפרמטר משתנה מטיפוס Ray המייצג את הקרן היוצאת מהמצלמה. היא עוברת על כל הצורות גיאומטריות שבסצנה (ע"י איטרטור של רשימת הצורות הגיאומטריות) ובודקת האם יש להן נקודות חיתוך עם הקרן (בעזרת הפונקציה findIntersections הממומשת ב-Geometryשמוצאת נקודות חיתוך בין הקרן לצורה). הפונקציה מחזירה רשימה מטיפוס <<Map<Geometry, List<Point3D של כל נקודות החיתוך של כל הצורות הגיאומטריות בסצנה עם הקרן הנתונה.

הפונקציה **getClosestPoint** מקבלת כפרמטר משתנה מטיפוס <<Map<Geometry, List<Point3D, דהיינו מפה של צורות גיאומטריות ורשימה של נקודות החיתוך של כל צורה, ומוצאת את הנקודה הכי קרובה למצלמה. היא עוברת על כל הנקודות ובודקת את מרחקה ע"י הפונקציה distance שבמחלקה Point3D. כל פעם שמוצאים נקודה יותר קרובה, הערכים של הנקודה והצורה נכנסים לתוך המשתנה minDistancePoint שהוא מטיפוס <<Map.Entry<Geometry, List<Point3D . בסוף התהליך, הפונקציה מחזירה את המשתנה minDistancePoint בו הערך של הנקודה הקרובה ביותר עם הצורה הגיאומטרית של אותה נקודה. חשוב להחזיר גם את הצורה כיוון שלכל צורה יש את החומר והצבע הייחודי לה ויש להתחשב בכך בסוף כאשר אנו רואים זאת במסך.

הפונקציה **printGrid** יוצרת רשת. היא מקבלת כפרמטר משתנה מסוג int המייצג את המרווח של הרשת. הפונקציה עוברת על המסך בקפיצות ע"פ הפרמטר שקיבלה וצובעת את הרשת ע"פ הצבע המתאים גם לאורך וגם לרוחב. הנתונים נכנסים למשתנה imageWriter_.

הפונקציה **writeToImage** קוראת לפונקציה המתאימה שבמחלקה ImageWriter , המדפיסה את הנתונים שבמשתנה imageWriter למסך.

הפונקציה **calcColor** אחראית על הצבע הסופי הניתן לכל נקודה. הפונקציה מקבלת משתנה Geometry המייצג צורה גיאומטרית, משתנה מטיפוס Point3D המייצג את הנקודה הרצויה, וקרן. היא מחזירה ערך מטיפוס Color שהוא הצבע הסופי שהתקבל בנקודה . הפונקציה בנויה לפי מודל פונג שנתן מספר פרמטרים לפיהם יוצג הצבע המתאים:

א. אור סביבתי (Ambient Light) .

ב. הצבע והחומר של העצם.

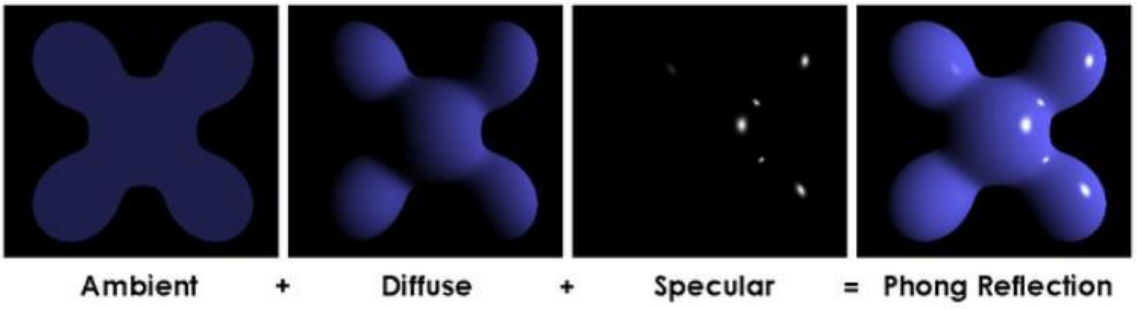
ג. הפיזור של האור - diffuseLight (בהתחשב בכל גופי התאורה בסצנה).

ד. הברק של האור - specularLight (בהתחשב בכל גופי התאורה בסצנה).

הפונקציה בנויה לפי האלגוריתם הבא:

קלוט למשתנה את הצבע של התאורה הכללית (אור סביבתי) ע"י הפונקציה getIntersity (במחלקה AmbientLight), קלוט את הצבע שהצורה עצמה מפיקה ע"י הפונקציה getEmmission (במחלקה Geometry), עבור על כל התאורות בסצנה וחשב את הפיזור של האור בנקודה הרצויה של כל אור ע"י הפונקציה calcDiffusiveComp , וכנ"ל לגבי הברק ע"י הפונקציה calcSpecularComp . חבר את כל הנתונים והחזר את הצבע הנוצר.

להמחשה:



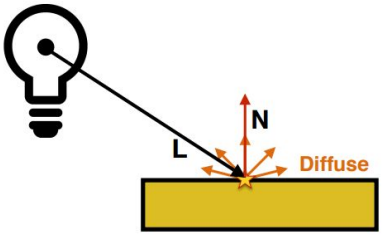
הפונקציה calcDiffusiveComp אחראית על החזרת צבע הפיזור בנקודה מסויימת. הסבר על הפרמטרים שהיא מקבלת:

Kd משתנה double המייצג פרמטר קבוע שמכפיל את האור בהתאם לכל צורה.

Normal משתנה Vector המייצג נורמל של הצורה הנתונה בנקודה.

L משתנה Vector המייצג וקטור של הכיוון של התאורה.

LightIntensity משתנה Color המייצג את צבע התאורה עצמה. איור ממחיש:



$$I_{point3D} = I_E + K_{AM}I_{AM} + \boxed{K_D(N \cdot L)I_L}$$

הסבר: הפונקציה מכפילה את המשתנה kd , עם התוצאה של המכפלה הסקלרית בין הנורמל לבין L (כיוון התאורה) שזה מבטא את הזווית של הכיוון לעומת הנורמל וכך הפיזור יותר גדול. כל זה מוכפל בצבע התאורה

עצמה של גוף התאורה (אם זה פרוז'קטור או נורה). כאשר מכפילים את התוצאה, הכוונה שלוקחים את הערכים ה- int ים של - RGB ומכפילים בערכים של ה- RGB השני. כמובן אם הערך עובר את 255 הוא מאותחל ל- 255 . הפונקציה calcSpecularComp מחזירה את הברק של הצורה הגיאומטרית בנקודה מסויימת לפי התאורה.

הסבר הפרמטרים:

Ks משתנה מטיפוס – double המייצג פרמטר קבוע שמכפיל את האור בהתאם לכל צורה.

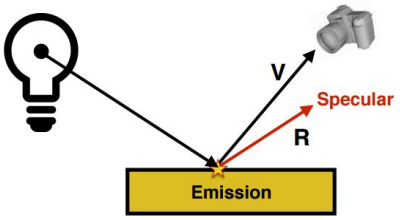
V משתנה מטיפוס – Vector המייצג וקטור אנכי לכיוון הקרן של התאורה לצורה הגיאומטרית.

Normal משתנה מטיפוס – Vector המייצג נורמל של הצורה הנתונה בנקודה.

L משתנה מטיפוס – Vector המייצג וקטור של הכיוון של התאורה.

Shininess משתנה מטיפוס – double המייצג את הזוהר של הנקודה. (יעלה נתון מסויים בחזקה)

LightIntensity משתנה מטיפוס – Color המייצג את צבע התאורה עצמה. איור ממחיש:



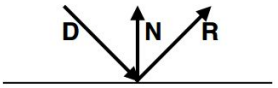
$$I_{point3D} = I_E + K_{AM}I_{AM} + K_D(N \cdot L)I_L + \boxed{K_S(V \cdot R)^n I_L}$$

הפונקציה מחשבת את המסגרת האדומה. המכפלה הסקלרית בין V לבין R, מורה על הזווית בין המצלמה לקרן

החוזרת מהאור (זה בעצם מידול של העולם הפיזיקאלי של משהו מבריק = קרן החזרת לעין מהתאורה).

להלן חישוב הפרמטר R :

Calculating R



$$R = D - 2(D \cdot N)N$$

התוצאה עולה בחזקת ה- Shininess וזאת מכיוון שבעולם הפיזיקאלי אור הברק יורד בצורה מאוד מהירה. כל זה מוכפל כמובן באור. כאשר מכפילים את התוצאה, הכוונה שלוקחים את הערכים ה- int ים של - RGB ומכפילים בערכים של ה-RGB השני. כמובן אם הערך עובר את 255 הוא מאותחל ל- 255 . (כמו לעיל בפונקציה calcDiffusiveComp).

שלב ד - שיפורי תמונה וביצועים:

1. ריבוי קרני החזרה (Multiple reflection rays) וריבוי קרני בליעה (Multiple refraction rays) במשטחים מבריקים.

בעולם האמיתי ישנם חומרים שאמנם מבריקים או שקופים אך יוצרים טשטוש של מה שנראה דרכם, ובמיוחד טשטוש של מה שרחוק מהם. כדי ליצור אפקט זה, במקום ליצור קרן השתקפות (וכנ"ל שבירה) אחת, ניצור מספר קרני השתקפות אקראיים בתוך מעגל סביב הקרן המקורית, נאסוף את הצבע שמגיע מכל הקרניים ונחלק במספר הקרניים.

זה יוצר את אפקט הטשטוש במידה טובה כתלות בזווית ההיסט של הקרניים וכך רמת הטשטוש של האובייקט המשתקף תלויה במרחקו מהמראה.

תכונה זו תלויה בחומר ולכן יצרנו פרמטר **blurring** שאותו הוספנו למחלקה **material**.

2. בעיה: קצוות משוננים. פתרון: ריבוי קרניים דרך הפיקסלים - Multiple eye rays.

את הפתרון לבעיה זו מימשנו באמצעות ריבוי קרני בליעה. (Multiple refraction rays), שמימשנו לעייל, ע"י כך שהוספנו לסצנה מישור (Plain) שקוף עם ערך מתאים בפרמטר blurring כך שישלח קרני שבירה מרובים דרכו. (הערה: אין כאן בעיית ביצועים של שליחת קרני שבירה אחורה, בזכות השיפור שעשינו שבו לא שולחים קרני השתקפות מחומר שאינו מבריק). זהו פתרון קומפקטי שאינו פוגע בביצועים או בתוצאה.

3. שיפור ביצועים ע"י 'קיצוץ' חלק מהרקורסיה:

הבעיה: אנחנו שולחים קרני רקורסיה גם כאשר את הצבע שנקבל נכפיל באפס או במקדם נמוך מאוד ולא תהיה השפעה לתוספת הצבע. למשל כאשר 'עוברים' דרך אובייקטים אטומים, או השתקפות של חומר שאינו מבריק כלל, או שכעוברים דרך כמה גיאומטריות חצי שקופות, ובמצטבר לאחר כמה דרגות כבר לא רואים כמעט כלום.

התוצאה היא חיסכון בזמן ריצה של מאות אחוזים(!) בסצנות מסויימות (עם רמת רקורסיה 6).

4. **הוספת המחלקה לייצוג קערה - Hemisphere (חצי כדור)**

המחלקה מייצגת קערה - חצי כדור במרחב.מחלקה זו מאפשרת לייצג צורה מורכבת יותר. כאן היינו צריכים להתמודד עם החזרת הצבע הפנימי והחיצוני באמצעות נורמל מתאים, וכן יצירת צל שהקערה עושה על עצמה, לפי הצורך. עיין בפירוט בפרק שעוסק בגיאומטריות.

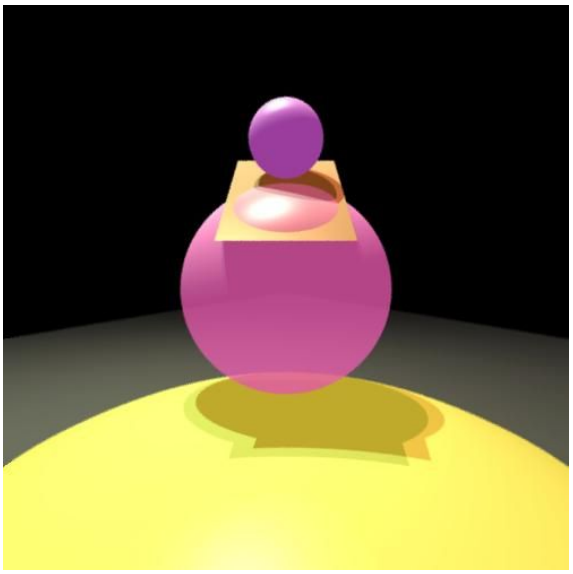
5. שינוי פונקציית הממשק GetNormal כך שתקבל ווקטור כיוון.

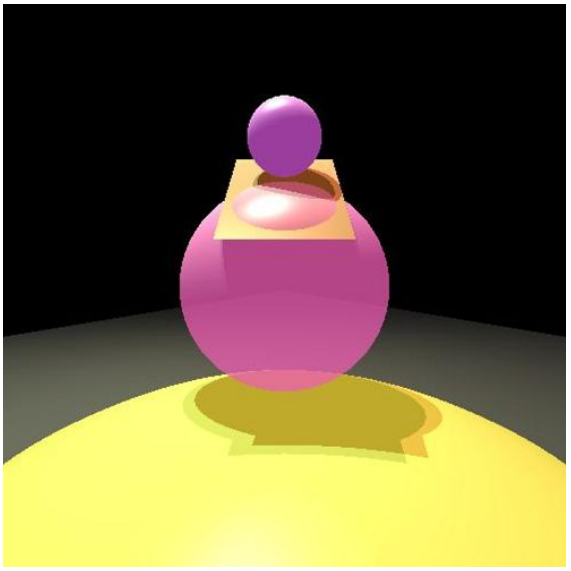
הבעיה: בפרוייקט עד היום, כיוון הנורמל היה לא דטרמיניסטי. לצורך חישובים שונים אנחנו משתמשים בנורמל (למשל הוספת אפסילון לכיוון הנכון בקרני שבירה והשתקפות, חישוב מכפלה סקלרית במודל phong) וכן עד היום sphare יצר רק קרניים החוצה וזה כבר לא מספיק כשיש כדורים חצי שקופים.

ההתמודדות עם בעיה זו עד היום הייתה באמצעים שונים שאינם מספקים ומסרבלים את הקוד. (ערך מוחלט בחישוב מכפלה סקלרית, בדיקה 'ידנית' של הכיוון, או בדיקה האם מדובר בflat geometry) מהסיבות הנ"ל החלטנו לשנות את אופן פעולות הפונקציה GetNormal ושמנו לב שתמיד כשתשתמשים בה יש ווקטור שלפיו אנחנו יודעים לאיזה כיוון אנחנו רוצים את הנורמל. לכן החתימה החדשה של הפונקציה היא:

```
public abstract Vector getNormal(Point3D point, Vector direction);
```

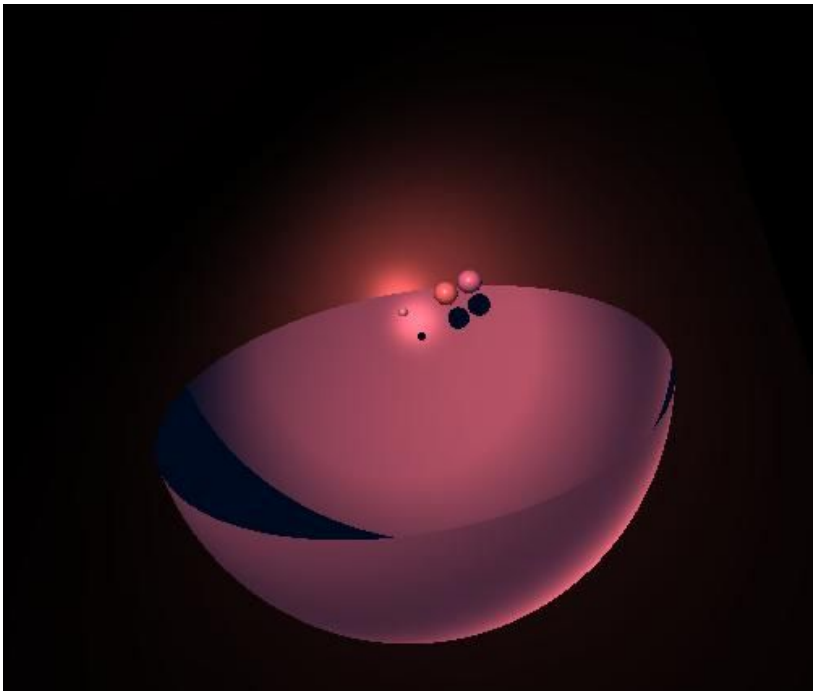
כך שתמיד יוחזר הנורמל לכיוון שממנו בדיע הווקטור direction.

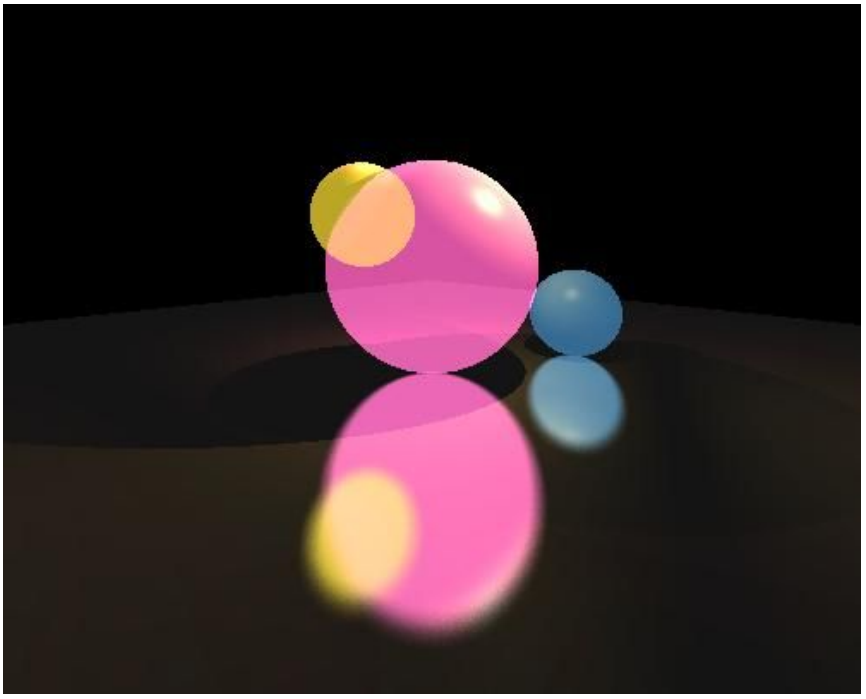


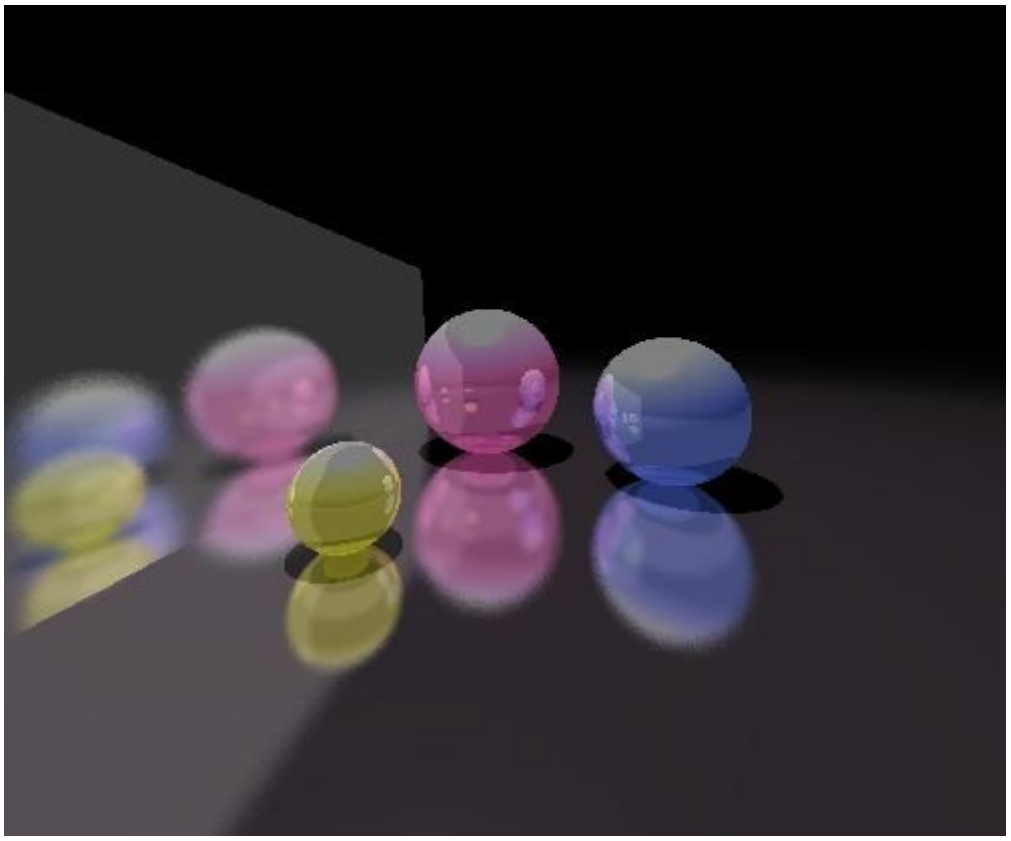


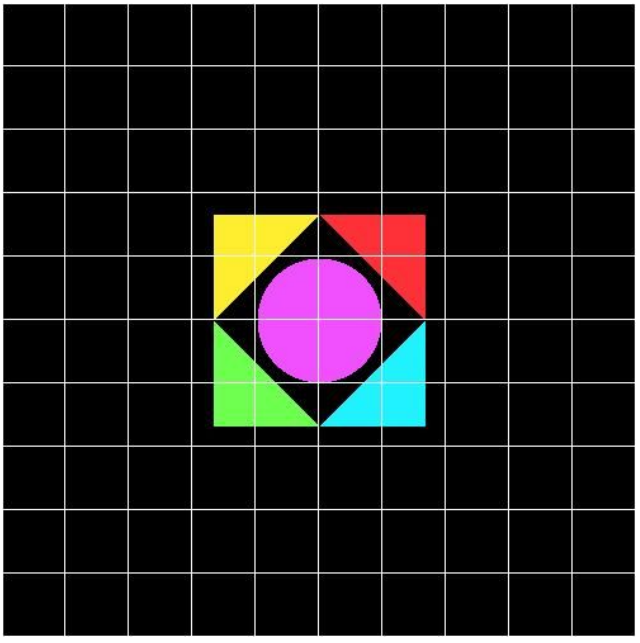
לפני:

אחרי:

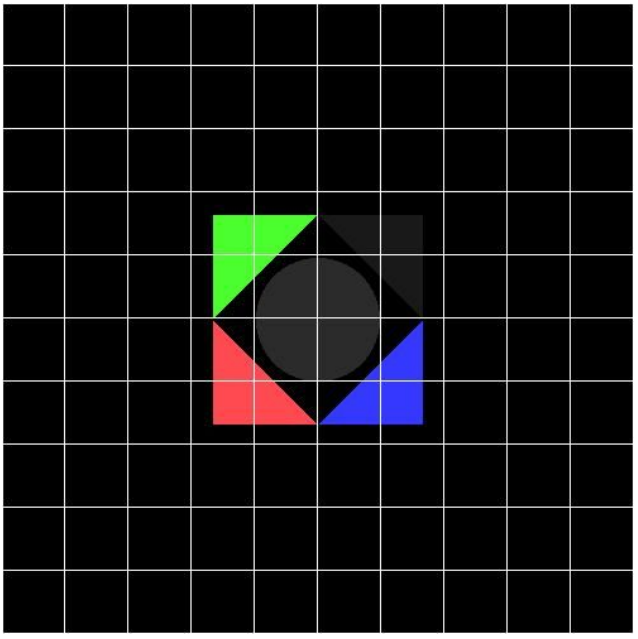




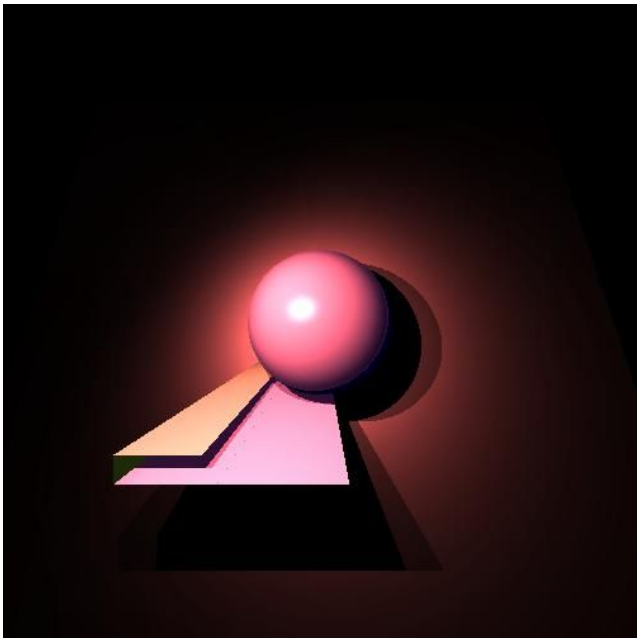




שימוש ב AmbientLight:



רקורסיה:



רקורסיה עם שקיפות והשתקפות: