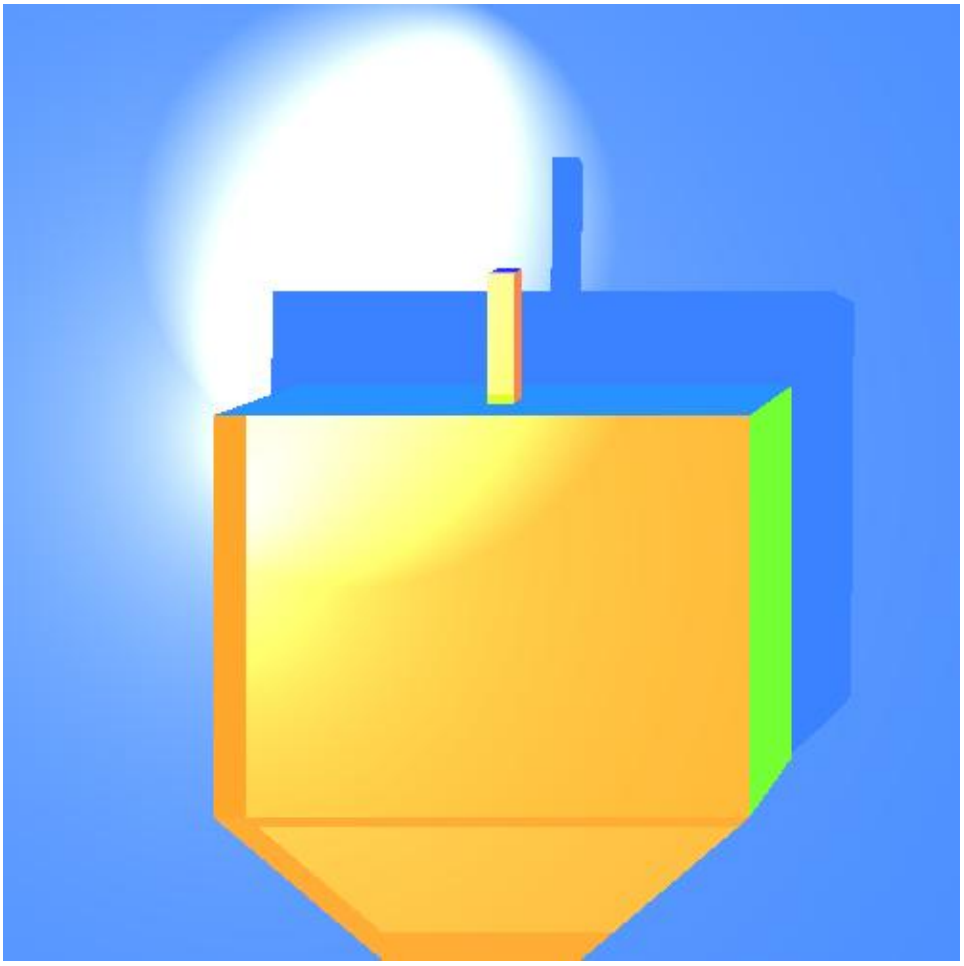
מיני פרויקט במבוא להנדסת תוכנה



Aron Scemama 337596274
Yaacov Elbaz 342492436
David Levy 342492956

תוכן עניינים

מבוא

אחד הנושאים החשובים והמאתגרים כיום בשוק המחשבים הוא הגרפיקה. בפרויקט זה אנו נתנסה בהבנה ותכנות של אובייקטים פשוטים במרחב. אנו נצור מודל המדמה צורות בסיסיות במרחב שלנו.
המיני פרויקט תוכנת בשפת JAVA. (לבקשת הקורס)
ברצוננו להצליח להקרין על המסך תמונה גרפית הנראית דומה למציאות של העולם שלנו.
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.

תחילה **נתאר לפי איזה כללים תכנתנו** את המיני פרויקט.

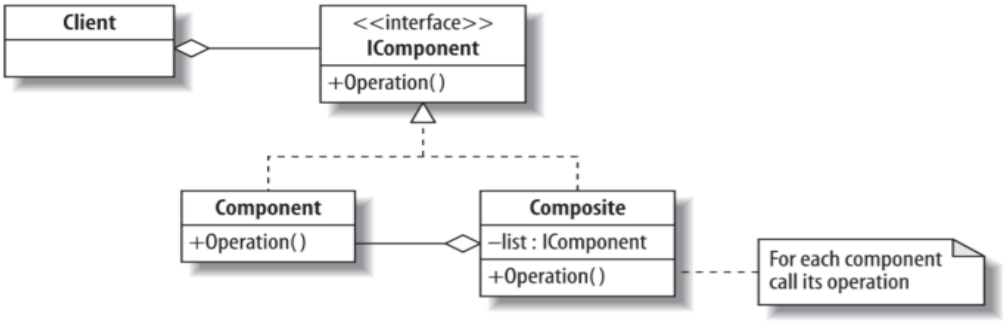
א. תכנות בשיטת agile.
ב. תכנות בסיסי ללא שימוש בספריות גרפיקה מוכנות.
ג. הבנה של העולם הפיזיקאלי ומידולו.
ד. תיעוד מפורט.

הסבר:

א. <u>תכנות בשיטת agile</u> - לאור הביקוש הרב כיום להתחדשות וקידמה, יש יתרון גדול לתכנות בשיטה זו. שיטה זו מיועדת לעבודה מהירה ונקייה, וכל זה בכדי להוציא מוצרים מוגמרים במהירות עם יכולת שדרוג מתמיד. נביא פה את עקרונות השיטה:

1. ראש סדר העדיפויות הוא לספק מוקדם ובאופן רציף ככל האפשר תוכנה בעלת ערך ללקוח.
2. מצפים בברכה לשינויים בדרישות התוכנה, אפילו בשלב מתקדם של הפיתוח. תהליכים מהירים ברי-שינוי מאפשרים יתרון משמעותי בתחרות הלקוח.
3. אספקה של תוכנה עובדת באופן תדיר, ממספר שבועות עד למספר חודשים, עם שאיפה לסולם זמנים קצר.
4. תהליכים "קלי תנועה" (sustainable development) מקדמים תהליך שאינו פוגע בעתיד על מנת ליהנות ממנו בהווה.
5. יש לספק תשומת לב מתמשכת למצוינות טכנולוגית ועיצוב (Design Patterns) טוב שמגביר את "קלות התנועה".
6. פשטות - היא חיונית (האומנות למקסם את כמות העבודה שלא תעשה).
7. הארכיטקטורה, העיצוב והדרישות חייבות להתבצע בתוך "קבוצות עצמאיות" (כמו עוצבה בצבא שמנהלת את עצמה ובעלת כל המשאבים).

נבאר בקצרה את ה-Design Patterns שזהו הדבר הכי חשוב בפרויקט זה. בכדי לשמר את המוצר ולעשות אותו רך ואפשרי לשינוי הקפדנו על מספר דברים: עבדנו ע"פ עיקרון תכנות מונחה עצמים, הקפדנו לעשות עיצוב בצורה תבניתית הנותן כוח למתכנת להתייחס לכמה אובייקטים ע"י הפעלת פונקציה דומה. דבר זה נותן כוח רב לתוכנה לשמר את עצמה, ולהיות בעלת יכולת גבוהה לשינויים סביבתיים.
דוגמה:



ב. <u>תכנות בסיסי ללא שימוש בספריות גרפיקה מוכנות</u> - בכדי לחוש ולהבין את הכללים הפיזיקאליים שעוזרים לנו לייצר סצנה גראפית, עבדנו החל מהבסיס ע"י ייצוג של קואורדינאטה במרחב מטיפוסים של double. לאחר מכן התקדמנו לנקודה, לצורות בסיסיות, וכו'....
ג. <u>הבנה של העולם הפיזיקאלי ומידולו</u> - עבדנו על הבנה של העולם הפיזיקאלי הכולל. יצירת צורות מרכבות ע"י צורות פשוטות כמו כדור ומשולש היוצרים את כל הצורות שברצוננו לייצג. הבנה של תאורות שונות שאנו נפגשים איתם בחיי היום יום. הבנה של הריאקציה שהתאורה מפיקה על עצם במרחב. לאחר ההבנה הזו יש צורך ליישם את זה בתוכנה ע"י חשיבה כיצד למדל וליצור עצם שנראה דומה לעצם במציאות שלנו מצד אחד, ומצד שני שיהיה מספיק פשוט לחישוב ולהצגה של הדברים במסך המחשב של המשתמש.
ד. <u>תיעוד מפורט</u> - בפרויקט, כחלק מעקרונות השיטה של agile, הקפדנו על תיעוד מפורט בכדי שהקוד יהיה קריא וניתן להבנה בקלות. התיעוד נעשה בפורמט זהה לכל הפונקציות:

```
/*****************************************************
* FUNCTION
*   genPrime
* PARAMETERS
*   int – highest possible prime value
* RETURN VALUE
*   A (positive) integer: the highest prime number,
*   smaller than the integer received as parameter.
* MEANING
*   This functions computes a prime number p, such that
*             0 <= p <= PARAMETER
* SEE ALSO
*   list of names of other functions in your system,
*   related to this function.
*****************************************************/
```

בנוסף יש את חוברת התיעוד הזו המפרטת את הפרויקט בפרוטרוט.

מבנה המיני פרויקט:

כל "נושא" חולק לחבילה משלו כחלק מהעיצוב ומתכנות מונחה עצמים.
כל נושא נתמך ומשתמש בעיקרון של תכנות מונחה עצמים כייעול המערכת ומיעוט בכפילויות של קוד.
אלו הנושאים:

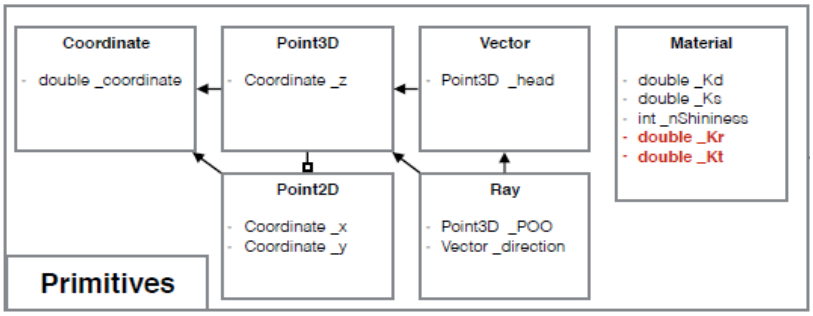
א. Primitives - מייצג את האובייקטים הגיאומטריים הפשוטים ביותר. התחלנו מקואורדינאטה, עברנו לנקודה במישור, במרחב, עד לייצוג וקטור וקרן. בכך נתנו את המסגרת לייצוג של צורות.
ב. Geometries - מייצג צורות גיאומטריות פשוטות. לדוגמה משולש, כדור...
ג. Elements - מייצג את המצלמה, דהינו נקודת המבט של הסצנה, וכן מיצג את "אביזרי" האור בהם אנו משתמשים בסצנה. (אור קבוע, שמש, נורה, פרוז'קטור....)
ד. Scene - מייצג את מכלול הסצנה ע"י ריכוז הצורות הגיאומטריות בתמונה, התאורה, הרקע וכו'...
ה. Renderer - לבסוף הצגת התמונה למשתמש ע"י זיהוי צבע מדויק לכל נקודה בסצנה. תהליך זה דורש חישובים רבים כמו מציאת נקודות המפגש של הצורות הגיאומטריות עם הנקודה הנידונה, מציאת הנקודה הקרובה ביותר. מציאת הצבע המתאים ע"פ חישוב התאורה והריאקציה בינה לעצם ע"פ סוגו ומיקומו במרחב.

חבילות הפרויקט

חבילת Primitives:

חבילה המייצגת את האובייקטים הגיאומטריים הפשוטים ביותר.

מבנה החבילה:



מחלקת Coordinate:

המחלקה מייצגת קואורדינאטה אחת. ישנו משתנה אחד למחלקה מטיפוס double בעל הרשאת private, המייצג ערך של קואורדינאטה על ציר מסוים. מבנה המחלקה:

```
private double _coordinate;
// ***************** Constructors ********************** //
public Coordinate();
public Coordinate(double coordinate);
public Coordinate(Coordinate coordinate)
// ***************** Getters/Setters ********************** //
public double getCoordinate();
public void setCoordinate(double coordinate);
// ***************** Administration ******************** //
public int compareTo(Coordinate coordinate);
// ***************** Operations ******************** //
public void add (Coordinate coordinate);
public void subtract (Coordinate coordinate);
```

למחלקה יש שלושה בנאים:

1) בנאי ברירת מחדל שמגדיר את הקואורדינאטה בערך 0.

2) בנאי שמקבל ערך מטיפוס double ומכניס את ערכו למשתנה המקומי של המחלקה.

3) בנאי העתקה.

הפונקציות **get/set** מקבלות ומחזירות את הערך של המשתנה המקומי.

הפונקציה **compareTo** בודקת האם הקואורדינטה המקומית שווה לקואורדינטה המתקבלת כפרמטר ע"י השוואת שני המשתנים שלהם שהם מטיפוס double. אם הם שווים היא מחזירה 0 ואם לא היא מחזירה 1.

הפונקציה **add** מקבלת כפרמטר משתנה מטיפוס Coordinate ומכניסה למשתנה המקומי את חיבור הערכים מטיפוס double של שתי הקואורדינטות.

הפונקציה **subtract** מקבלת כפרמטר משתנה מטיפוס Coordinate ומכניסה למשתנה המקומי את חיסור הערכים בין המשתנה המקומי לבין המשתנה בפרמטר.

מחלקת Point2D:

המחלקה מייצגת נקודה במישור שמיוצג באמצעות ציר x וציר y. למחלקה יש שני משתנים מטיפוס Coordinate בעלי הרשאת protected המייצגים את מיקום הנקודה ביחס לציר x וציר y. מבנה המחלקה:

```
protected Coordinate _x;
protected Coordinate _y;
// ***************** Constructors ********************** //
public Point2D();
public Point2D(Coordinate x, Coordinate y);
public Point2D(Point2D point2D);
// ***************** Getters/Setters ********************** //
public Coordinate getX();
public Coordinate getY();
public void setX(Coordinate _x);
public void setY(Coordinate _y);
// ***************** Administration ******************** //
public int compareTo(Point2D point2D);
```

למחלקה יש שלושה בנאים:

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

2) בנאי שמקבל שני ערכים מטיפוס Coordinate ומכניס את ערכם למשתנים המקומיים של המחלקה.

3) בנאי העתקה.

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

הפונקציה **compareTo** בודקת האם הנקודה המקומית שווה לנקודה המתקבלת כפרמטר ע"י השוואת ערכי הקואורדינטות שלהם. אם שתי הקואורדינטות שוות בשתי הנקודות היא מחזירה 0 ואם לא היא מחזירה 1.

מחלקת Point3D:

המחלקה מייצגת נקודה במרחב שמיוצג באמצעות ציר x, ציר y וציר z. המחלקה יורשת ממחלקת Point2D. למחלקה יש משתנה אחד מטיפוס Coordinate בעל הרשאת private המייצג את מיקום הנקודה ביחס לציר z. מבנה המחלקה:

```
private Coordinate _z;
// ***************** Constructors ********************** //
public Point3D();
public Point3D(Coordinate x, Coordinate y, Coordinate z);
public Point3D(double x, double y, double z);
public Point3D(Point3D point3D);
// ***************** Getters/Setters ********************** //
public Coordinate getZ();
public void setZ(Coordinate _z);
// ***************** Administration ******************** //
public int compareTo(Point3D point3D);
```

```
public String toString();
// ***************** Operations ******************** //
public void add(Vector vector);
public void subtract(Vector vector);
public double distance(Point3D point);
```

למחלקה יש ארבעה בנאים:

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

2) בנאי שמקבל שלושה ערכים מטיפוס Coordinate ומכניס את ערכם למשתנים המקומיים של המחלקה.

3) בנאי שמקבל שלושה ערכים מטיפוס double ומכניס את ערכם לקואורדינאטות של המחלקה.

4) בנאי העתקה.

הפונקציות **get/set** מקבלות ומחזירות את הערך של המשתנה המקומי.

הפונקציה **compareTo** בודקת האם הנקודה המקומית שווה לנקודה המתקבלת כפרמטר ע"י השוואת ערכי הקואורדינטות שלהם. אם שלושת הקואורדינטות שוות בשתי הנקודות היא מחזירה 0 ואם לא היא מחזירה 1.

הפונקציה **toString** מדפיסה את ערך הנקודה בצורה זו: (x, y, z). (עד שתי ספרות לאחר הנקודה העשרונית)

הפונקציה **add** מקבלת כפרמטר משתנה מטיפוס Vector ומכניסה למשתנים המקומיים את חיבור הקואורדינטות של הנקודה המקומית ושל הוקטור.

הפונקציה **subtract** מקבלת כפרמטר משתנה מטיפוס Vector ומכניסה למשתנים המקומיים את חיסור הקואורדינטות בין הנקודה המקומית לבין הקואורדינטות שבוקטור.

הפונקציה **distance** מקבלת כפרמטר משתנה מטיפוס Point3D ומחזירה ערך מטיפוס double המייצג את המרחק בין הפרמטר לבין הנקודה המקומית ע"י שימוש במשפט פיתגורס במרחב.

מחלקת Vector:

המחלקה מייצגת וקטור במרחב. למחלקה יש משתנה אחד מטיפוס Point3D בעל הרשאת private המייצג את ראש הוקטור מראשית הצירים. מבנה המחלקה:

```
private Point3D _head;
// ***************** Constructors ********************** //
public Vector();
public Vector(Point3D head);
public Vector(Vector vector);
public Vector(double xHead, double yHead, double zHead);
public Vector(Point3D p1, Point3D p2);
// ***************** Getters/Setters ********************** //
public Point3D getHead()
public void setHead(Point3D head);
// ***************** Administration ******************** //
public int compareTo(Vector vector);
public String toString();
// ***************** Operations ******************** //
public void add (Vector vector );
public void subtract (Vector vector);
public void scale(double scalingFactor);
```

```
public Vector crossProduct(Vector vector);
public double length();
public void normalize(); // Throws exception if length = 0
public double dotProduct(Vector vector);
```

למחלקה יש חמישה בנאים:

1) בנאי ברירת מחדל שמגדיר את ראש הוקטור בערך (0,0,0).

2) בנאי שמקבל ערך מטיפוס Point3D ומכניס את ערכו למשתנה המקומי של המחלקה.

3) בנאי העתקה.

4) בנאי שמקבל שלושה ערכים מטיפוס double ומכניס את ערכם למשתנים של הנקודה של המשתנה המקומי.

5) בנאי שמקבל שני ערכים מטיפוס Point3D ומכניס את ערך החיסור ביניהם למשתנה המקומי של המחלקה.

הפונקציות **get/set** מקבלות ומחזירות את הערך של המשתנה המקומי.

הפונקציה **compareTo** בודקת האם ראש הוקטור המקומי שווה לנקודה המתקבלת כפרמטר ע"י השוואת ערכי הנקודות שלהם. אם הן שוות היא מחזירה 0 ואם לא היא מחזירה 1.

הפונקציה **toString** מדפיסה את ערך ראש הוקטור כמו במחלקת Point3D.

הפונקציה **add** מקבלת כפרמטר משתנה מטיפוס Vector ומכניסה לראש הוקטור המקומי את חיבור הנקודות שלו ושל ראש הוקטור שהתקבל כפרמטר.

- Vector addition: $$u+v=\begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix}+\begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix}=\begin{pmatrix} u_1+v_1 \\ u_2+v_2 \\ u_3+v_3 \end{pmatrix}$$

הפונקציה **subtract** מקבלת כפרמטר משתנה מטיפוס Vector ומכניסה לראש הוקטור המקומי את חיסור הנקודות בינו לבין ראש הוקטור שהתקבל כפרמטר.

הפונקציה **scale** מקבלת כפרמטר משתנה מטיפוס double ומכניסה לוקטור המקומי את תוצאת הכפל בסקלר בין הוקטור ובין הסקלר שהתקבל כפרמטר. הוקטור באותו כיוון מתארך (אם הסקלר גדול מ-1), מתקצר (אם הסקלר קטן מ-1) או הופך כיוון (אם הסקלר שלילי) ואם הסקלר 1 הוקטור לא משתנה.

- Scalar multiplication: $$\alpha v=\alpha\begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix}=\begin{pmatrix} \alpha v_1 \\ \alpha v_2 \\ \alpha v_3 \end{pmatrix}$$

הפונקציה **crossProduct** מקבלת כפרמטר משתנה מטיפוס Vector ומחזירה ערך מטיפוס Vector המייצג את תוצאת המכפלה הוקטורית בין הוקטור המקומי ובין הוקטור שהתקבל כפרמטר. הוקטור המתקבל מאונך לשני הוקטורים הנתונים. אם הוקטורים מקבילים התוצאה תהיה וקטור 0.

- Cross product: $$u \times v = \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} \times \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} = \begin{pmatrix} u_2v_3 - u_3v_2 \\ u_3v_1 - u_1v_3 \\ u_1v_2 - u_2v_1 \end{pmatrix}$$

הפונקציה **length** מחזירה ערך מטיפוס double המייצג את אורך הוקטור מראשית הצירים ועד לראשו.

הפונקציה **normalize** מנרמלת את הוקטור המקומי, כלומר משנה את אורכו ל-1 באותו כיוון. הפונקציה מחזירה Exception אם מנסים לנרמל וקטור שאורכו הוא 0.

הפונקציה **dotProduct** מקבלת כפרמטר משתנה מטיפוס Vector ומחזירה ערך מטיפוס double המייצג את המכפלה הסקלרית בין הוקטור המקומי לבין הוקטור שהתקבל כפרמטר. התוצאה היא אורך ההיטל מהוקטור הראשון לשני. אם הוקטורים מאונכים התוצאה תהיה 0.

- Dot product: $$u \cdot v = u_1v_1 + u_2v_2 + u_3 + v_3$$

מחלקת Ray:

המחלקה מייצגת קרן במרחב. למחלקה יש שני משתנים, אחד מטיפוס Point3D המייצג את נקודת המקור של הקרן, ואחד מטיפוס Vector המייצג את כיוונה. שניהם בעלי הרשאת private. מבנה המחלקה:

```
private Point3D _POO;
private Vector _direction;
// ***************** Constructors ********************** //
public Ray();
public Ray(Ray ray);
public Ray(Point3D poo, Vector direction);
// ***************** Getters/Setters ********************** //
public void setPOO(Point3D _POO);
public void setDirection(Vector _direction);
public Vector getDirection();
public Point3D getPOO();
```

למחלקה יש שלושה בנאים:

1) בנאי ברירת מחדל שמגדיר את נקודת המקור ואת וקטור הכיוון בערך (0,0,0).

2) בנאי העתקה.

3) בנאי שמקבל שני ערכים, נקודת מקור מטיפוס Point3D וכיוון מטיפוס Vector, ומכניס את ערכם למשתנים המקומיים של המחלקה.

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

מחלקת Material:

המחלקה מייצג סוג של חומר ע"פ מדדים פיזיקאליים. למחלקה יש חמישה משתנים מטיפוס double בעלי הרשאת private המייצגים את תכונות החומר. מבנה המחלקה:

```
private double _Kd; // Diffusion attenuation coefficient
private double _Ks; // Specular attenuation coefficient
private double _Kr; // Reflection coefficient (1 for mirror)
private double _Kt; // Refraction coefficient (1 for transparent)
private double _n; // Refraction index
// ***************** Constructors ********************** //
public Material()
{
_Kd = 1;
_Ks = 1;
_Kr = 0;
_Kt = 0;
_n = 1;
}
public Material(Material material);
// ***************** Getters/Setters ********************** //
public double getKd();
public double getKs();
public double getKr();
public double getKt();
public double getN();
public void setKd(double _Kd);
public void setKs(double _Ks);
public void setKr(double _Kr);
public void setKt(double _Kt);
public void setN (double _n);
```

למחלקה יש שני בנאים:

1) בנאי ברירת מחדל שמגדיר את תכונות החומר כפי שמתואר במבנה דלעיל.

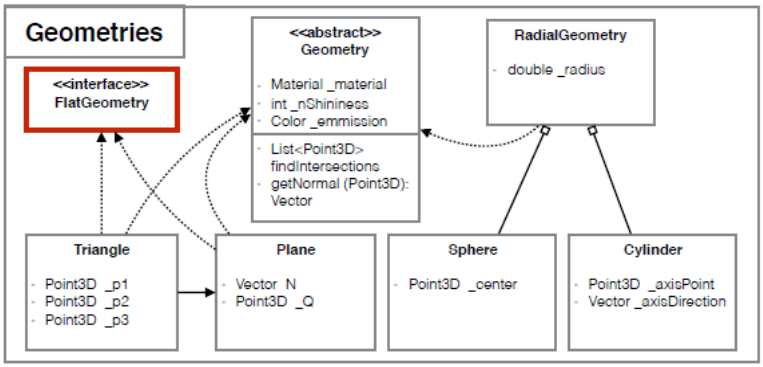
2) בנאי העתקה.

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

חבילת Geometries:

חבילה המייצגת צורות גיאומטריות פשוטות.

מבנה החבילה:



מחלקת Geometry:

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);
// ***************** Operations ******************** //
public abstract List<Point3D> FindIntersections (Ray ray);
public abstract Vector getNormal(Point3D point);
// ***************** Getters/Setters ********************** //
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);
```

הפונקציה **findIntersections** היא פונקציה אבסטרקטית המקבלת כפרמטר משתנה מטיפוס Ray ומחזירה רשימה בעלת טיפוסים Point3D המייצגים את נקודות החיתוך של הקרן עם הצורה הגיאומטרית.

הפונקציה **getNormal** היא פונקציה אבסטרקטית המקבלת כפרמטר משתנה מטיפוס Point3D ומחזירה Vector המייצג את הנורמל, כלומר מאונך לצורה הגיאומטרית בנקודה שהתקבלה כפרמטר.

הפונקציות **get/set** מקבלות ומחזירות את הערכים של המשתנים המקומיים ושל פרטי החומר של הצורה.

מחלקת RadialGeometry:

המחלקה מייצגת צורה גיאומטרית מעגלית. 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;
```

```
// ***************** Constructors ********************** //
public RadialGeometry();
public RadialGeometry(double radius);
// ***************** Getters/Setters ********************** //
public double getRadius();
public void setRadius(double radius);
```

למחלקה יש שני בנאים:

1) בנאי ברירת מחדל שמגדיר את הרדיוס באורך 0.

2) בנאי שמקבל ערך מטיפוס double ומכניס את ערכו לרדיוס המקומי של המחלקה.

הפונקציות **get/set** מקבלות ומחזירות את הערך של הרדיוס המקומי.

ממשק FlatGeometry:

הממשק מייצג סימון לצורות גיאומטריות שטוחות (כלומר שאינן מעגליות). אין לממשק זה פונקציות, אלא הוא בא רק לנוחות המימוש של הצורות הגיאומטריות.

מחלקת Plane:

המחלקה מייצגת מישור במרחב. המחלקה יורשת מהמחלקה Geometry וממשת כביכול 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);
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.

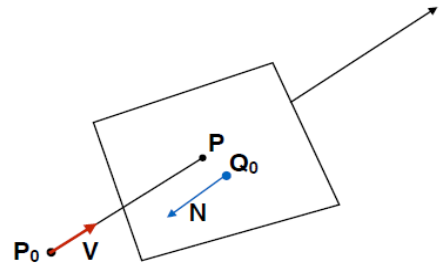
3) בנאי שמקבל שני ערכים, נורמל מטיפוס Vector ונקודה מטיפוס Point3D, ומכניס את ערכם למשתנים המקומיים של המחלקה.

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:

$$\text{Ray: } P = P_0 + tV$$
$$\text{Plane: } N \cdot (P - Q_0) = 0$$

$$N \cdot (P_0 + tV - Q_0) = 0$$
$$t = - N \cdot (P_0 - Q_0)/(N \cdot V)$$



מחלקת Triangle:

המחלקה מייצגת משולש במרחב. המחלקה יורשת מהמחלקה Geometry וממשת כביכול 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);
// ***************** 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);
public List<Point3D> FindIntersections(Ray ray);          private double getSign(Point3D poo,
Point3D p1, Point3D p2, Vector p_p0);
```

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.

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

הפונקציה **getNormal** מקבלת כפרמטר משתנה מטיפוס Point3D (במקרה הזה הוא לא רלוונטי) ומחזירה ערך מטיפוס Vector המייצג את הנורמל למשולש.

הפונקציה **findIntersections** מקבלת כפרמטר משתנה מטיפוס Ray ומחזירה רשימה בעלת טיפוסים 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.

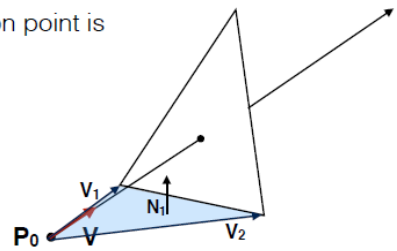
First, intersect ray with plane (normal is the cross product of the two sides of the triangle).

Then, check if the intersection point is inside triangle

For each side of the triangle:

V1=T1-P0
V2=T2-P0
N1=(V_2xV_1)/|V_2xV_1|

Then,
if sign((P-P0) · N1) ==
  sign((P-P0) · N2) ==
  sign((P-P0) · N3)
-> return true



הפונקציה **getSign** היא פונקצית עזר (ולכן היא בעלת הרשאת 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.

מחלקת Cylinder:

המחלקה מייצגת גליל במרחב. המחלקה יורשת ממחלקת RadialGeometry. למחלקה יש שני משתנים, אחד מטיפוס המייצג נקודה על הגליל ואחד מטיפוס Vector המייצג את כיוון הגליל. שניהם בעלי הרשאת private. 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 Vector getNormal(Point3D point);                public List<Point3D>
FindIntersections(Ray ray);
```

למחלקה יש שלושה בנאים:

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

2) בנאי העתקה.

3) בנאי שמקבל שלושה ערכים, רדיוס מטיפוס double, נקודה מטיפוס Point3D, וכיוון מטיפוס Vector ומכניס את ערכם למשתנים המקומיים של המחלקה.

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

הפונקציה **getNormal** מקבלת כפרמטר משתנה מטיפוס Point3D ומחזירה ערך מטיפוס Vector המייצג את הנורמל לגליל בנקודה שהתקבלה כפרמטר.

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

מחלקת Sphere:

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);
// ***************** Getters/Setters ********************** //
public Point3D getCenter();
public void setCenter(Point3D center);
// ***************** Operations ******************** //
public Vector getNormal(Point3D point);                    public List<Point3D>
FindIntersections(Ray ray);            private Point3D getPoint(double t, Vector v, Point3D poo);
```

למחלקה יש שלושה בנאים:

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

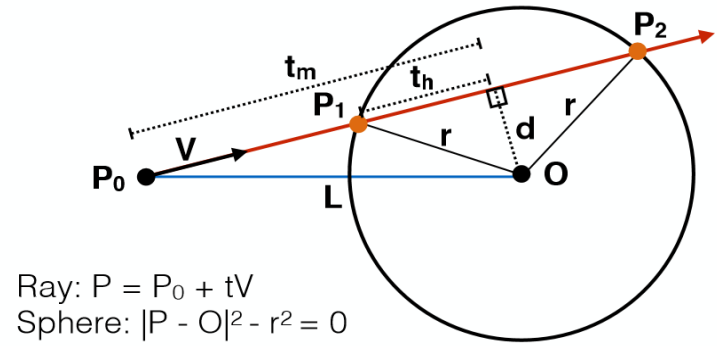
2) בנאי העתקה.

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

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

הפונקציה **getNormal** מקבלת כפרמטר משתנה מטיפוס Point3D ומחזירה ערך מטיפוס Vector המייצג את הנורמל לכדור בנקודה שהתקבלה כפרמטר.

הפונקציה **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. בכדי להבין את לוגיקת הפונקציה ישנו איור עזר.



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 והפונקציה תחזיר 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: 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.

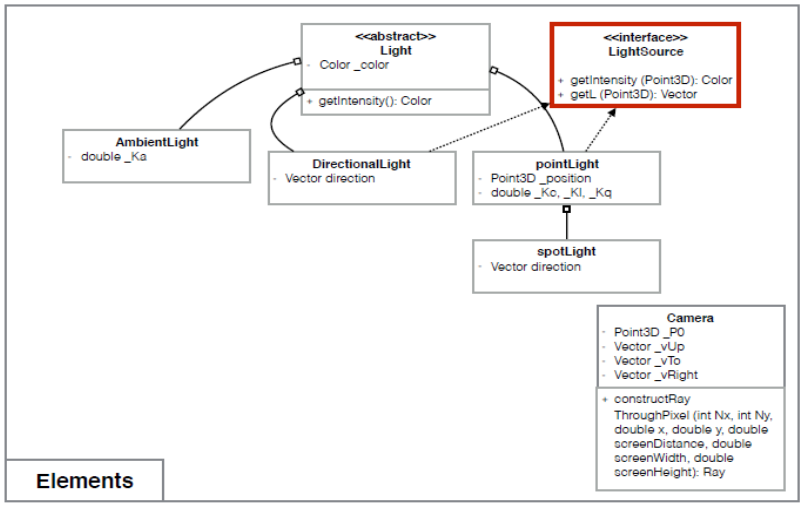
הפונקציה **getPoint** הינה פונקציה פנימית של המחלקה (ולכן היא בעלת הרשאת private) ונועדה לחשב את נקודת החיתוך עבור הכדור. הפונקציה נצרכת משום שפעמים יש שתי נקודות חיתוך וע"י אותו חישוב (עם פרמטרים שונים) ניתן למצוא אותם. הפונקציה מקבלת שלושה פרמטרים: t - המרחק בין הקרן המנורמלת לנקודת החיתוך. v - הכיוון של הקרן (מנורמל), poo – נקודת מיקום המצלמה. הפונקציה מחשבת את נקודת החיתוך ע"י הנוסחה הבאה:

P = P0 + tV (P0 הכוונה ל-poo), ומחזירה את הנקודה שנמצאה.

חבילת Elements:

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

מבנה החבילה:



מחלקת *Camera*:

המחלקה מייצגת מצלמה במרחב. למחלקה יש ארבעה משתנים, אחד מטיפוס Point3D המייצג את העין של המצלמה ושלושה מטיפוס Vector המייצגים את כיוון הצילום של המצלמה. ארבעתם בעלי הרשאת private. 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);
// ***************** 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 (0,0,0), 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.

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

הפונקציה **toString** מדפיסה את נתוני המצלמה בצורה זו:

Vto: (x, y, z)

Vup: (x, y, z)

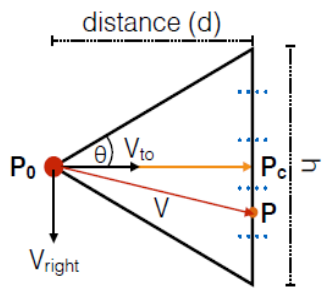
Vright: (x, y, z)

הפונקציה **constructRayThroughPixel** מקבלת כפרמטרים שבעה משתנים, שניים מטיפוס int המייצגים את מספר הפיקסלים במסך, וחמישה מטיפוס double 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)
$R_x = w/\#Pixels_x$
$R_y = h/\#Pixels_y$



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

מחלקת 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.

ממשק LightSource:

הממשק מייצג מקורות אור חיצוניים. מבנה הממשק:

```
public abstract Color getIntensity(Point3D point);
public abstract Vector getL(Point3D point); // light to point vector
```

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.

מחלקת AmbientLight:

המחלקה מייצגת אור סביבתי. המחלקה יורשת מהמחלקה Light. למחלקה יש 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(double r, double g, double b);
// ***************** 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.

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

הפונקציה **getIntensity** מחזירה ערך מסוג Color המייצג את עוצמת האור הסביבתי.

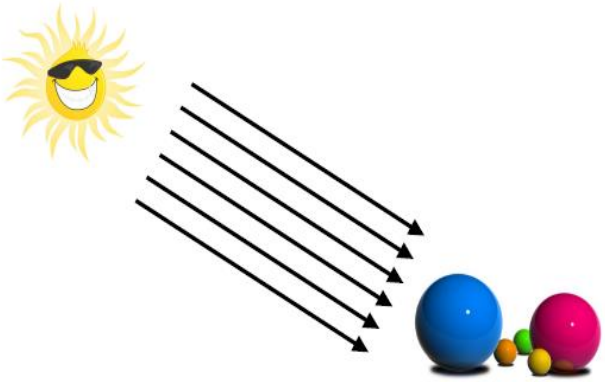
מחלקת DirectonalLight:

המחלקה מייצגת אור עם כיוון (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.



מחלקת PointLight:

המחלקה מייצגת נקודה המפיקה אור (כמו נורה). 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:

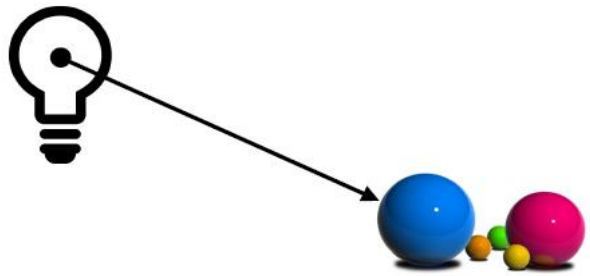
```
private Point3D _position;
private double _Kc, _Kl, _Kq;
// ***************** Constructors ********************** //
public PointLight(Color color, Point3D position,
double kc, double kl, double kq);
// ***************** Getters/Setters ********************** //
public Color getIntensity(Point3D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



מחלקת SpotLight:

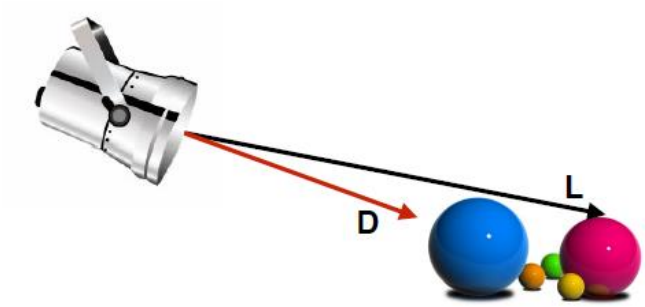
המחלקה מייצגת נקודה המפיקה אור עם כיוון (כמו פרוז'קטור). המחלקה יורשת מהמחלקה PointLight. למחלקה יש משתנה אחד מטיפוס Vector בעל הרשאת private, המייצג את כיוון האור. מבנה המחלקה:

```
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:

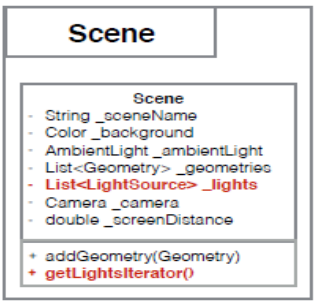
$$I_L = [I_0 \underbrace{\boxed{(D \cdot L)}}_{\text{dot product}}] / (K_c \cdot k_j d \cdot k_q d^2)$$



חבילת Scene:

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

מבנה החבילה:



מחלקת Scene:

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.

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

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

הפונקציה **getGeometriesIterator** מחזירה איטרטור (עוקב) מטיפוס Iterator<Geometry> לרשימת הצורות הגיאומטריות של הסצנה.

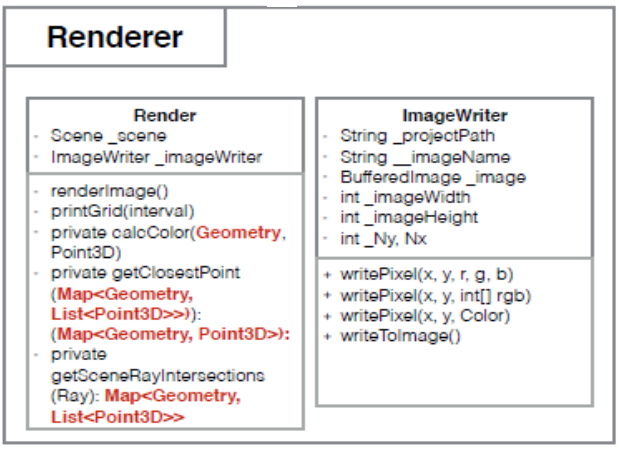
הפונקציה **addLight** מקבלת כפרמטר משתנה מטיפוס LightSource ומכניסה אותו לרשימת מקורות האור של הסצנה.

הפונקציה **getLightsIterator** מחזירה איטרטור (עוקב) מטיפוס Iterator<LightSource> לרשימת מקורות האור של הסצנה.

חבילת Renderer:

חבילה המייצגת את מחולל התמונה, המחשבת את הצבע המתאים לכל ומייצרת תמונה מתאימה.

מבנה החבילה:



מחלקת ImageWriter:

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.

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

הפונקציה **writeToImage** מייצרת את התמונה לתוך קובץ מסוג jpg.

הפונקציה **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.

מחלקת Render:

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> findClosestIntersection(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, מדפיס מטיפוס ImageWriter וסצנה מטיפוס 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.
- בכדי למצוא את הקרן משתמשים בפונקצית constructRayThroughPixel השייכת למחלקת המצלמה (עיין לעיל).
- קוראים לפונקציה findClosestIntersection שמחזירה את נקודת החיתוך הקרובה ביותר, אם קיימת. (הסבר על הפונקציה בהמשך)
- אם אין נקודות חיתוך, הכנס למשתנה imageWriter את הנקודה הנתונה ואת צבע הרקע.
- אם יש נקודת חיתוך אזי:
  - מצא את הצבע שצריך להיות בנקודה זו (הקרובה) ע"י הפונקציה calcColor (הסבר על הפונקציה בהמשך). אנו שולחים כפרמטרים את ה-key=Geometry ואת ה value=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-Geometry, שמוצאת 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<Geometry, 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 כיון 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 **addColors** היא פונקצית עזר המקבלת שני משתנים מטיפוס Color ומחזירה ערך מטיפוס Color המייצג את החיבור ביניהם. הפונקציה לוקחת את הערך ה-int של כל צבע (אדום, ירוק וכחול) ומחברת ביניהם. אם החיבור שווה יותר מ-255 אזי הפונקציה מאתחלת את הגורם ל-255 שהוא המספר המקסימלי האפשרי.

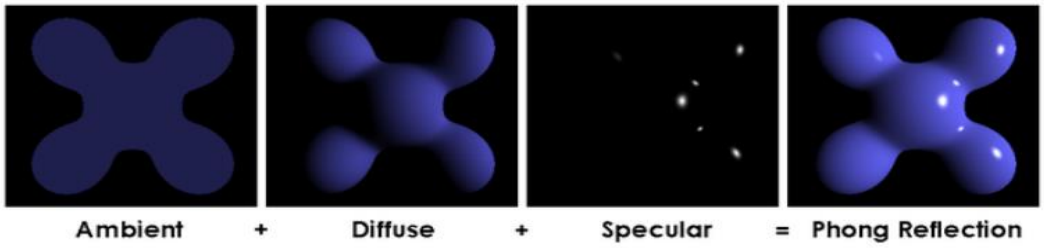
הפונקציה **calcColor** היא הפונקציה המורכבת ביותר בפרויקט.

באופן כללי: היא אחראית על הצבע הסופי הניתן לכל נקודה לפי ערכי פרמטרים שונים . פירוט: הפונקציה מקבלת שני פרמטרים, משתנה מטיפוס Geometry המייצג צורה גיאומטרית ומשתנה מטיפוס Point3D המייצג את הנקודה הרצויה. היא מחזירה ערך מטיפוס Color שהוא הצבע הסופי שהתקבל בנקודה . הפונקציה בנויה לפי שיטת 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 (קראנו לזה אור סביבתי) ע"י הפונקציה getIntersity (פונקציה במחלקת AmbientLight), קלוט את הצבע שהצורה עצמה מפיקה ע"י הפונקציה getEmmission (במחלקת 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. להלן מודל להמחשת האלגוריתם:



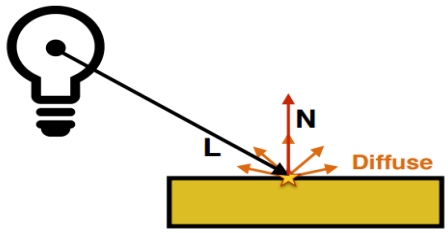
הפונקציה **calcDiffusiveComp** אחראית על החזרת צבע הפיזור בנקודה מסויימת. הסבר על הפרמטרים שהיא מקבלת:

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

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

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

LightIntensity – משתנה מטיפוס 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.

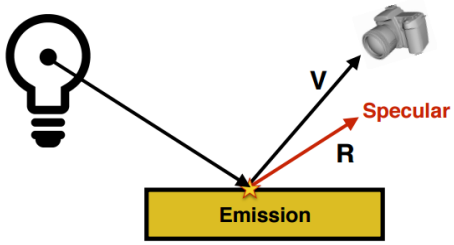
V – משתנה מטיפוס Vector המייצג וקטור אנכי לכיוון הקרן של התאורה לצורה הגיאומטרית.

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)

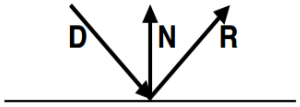
LightIntensity – משתנה מטיפוס Color המייצג את צבע התאורה עצמה. איור ממחיש:



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

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)