פיתוח תוכנת מחשב המדמה מערכת פיזיקלית של אופטיקה גיאומטרית



שם התלמיד: עומרי אלחי

בית הספר: תיכון אזורי מקיף צפית

יישוב: קיבוץ כפר מנחם

שם המנחה: ד"ר ולדימיר נודלמן

עיסוק המנחה: מרצה במכון הטכנולוגי חולון

שם המורה המלווה: ערן קצב

שנה: 2016 תשע"ו

**העבודה בוצעה כחלק מתוכנית "מדעי המחשב אקדמיה ותעשיה" של מכון דוידסון לחינוך מדעי**

# תוכן עניינים

[תוכן עניינים 2](#_Toc449191911)

[תקציר 3](#_Toc449191912)

[רקע כללי לנושא העבודה 3](#_Toc449191913)

[מטרת העבודה 4](#_Toc449191914)

[ניסוח וניתוח של בעיות הפיתוח והבעיה האלגורתמית 5](#_Toc449191915)

[תיאור אלגוריתמים ושיטות קיימות לפיתרון בעיות הפיתוח 6](#_Toc449191916)

[הפתרונות המוצעים 7](#_Toc449191917)

[תיאור המודולים של מערכת התוכנה 9](#_Toc449191918)

[מחלקות (עצמים) 9](#_Toc449191919)

[מחלקות שירות/עזר 10](#_Toc449191920)

[Windows Forms (מחלקה גרפית/חלונות) 11](#_Toc449191921)

[מבני נתונים שנעשה בהם שימוש 11](#_Toc449191922)

[סביבת עבודה 13](#_Toc449191923)

[הערכת פיתרון 14](#_Toc449191924)

[השוואה בין התוכנה לבין ישום קיים 14](#_Toc449191925)

[הבדלים בין התוכנה במצבה הנוכחי לתוכנה המתוכננת 15](#_Toc449191926)

[המלצות לשיפור התוכנה במצבה הנוכחי 15](#_Toc449191927)

[תיאור הממשק למשתמש 16](#_Toc449191928)

[תמונה מייצגת של ממשק המשתמש הראשי 20](#_Toc449191929)

[רפלקציה 21](#_Toc449191930)

[ביבליוגרפיה 22](#_Toc449191931)

[תודות 23](#_Toc449191932)

[נספחים 24](#_Toc449191933)

[דוגמאות מייצגות של הרצת המערכת 24](#_Toc449191934)

[תדפיס של הקוד המתועד 26](#_Toc449191935)

# תקציר

הפרויקט עוסק בפיתוח תוכנת מחשב, שמטרתה לדמות מערכת פיזיקלית הפועלת על אופטיקה גיאומטרית. אופטיקה גיאומטרית היא ענף בפיזיקה העוסק בתופעות האור שניתנות להתפרש כהתפשטות האור במרחב בקווים ישרים. בחרתי בנושא זה מתוך זיקה אישית לתחום הפיזיקה, ולתחום האופטיקה גיאומטרית בפרט. כמו כן מצאתי נושא זה הולם ומתאים למסגרת הפרויקט.

# רקע כללי לנושא העבודה

נושא הפרויקט הוא פיתוח תוכנת מחשב המדמה באופן חזותי ומילולי מערכת על בסיס התחום הפיזיקאלי אופטיקה גיאומטרית. הפרויקט יתבצע על ידי תהליך של פיתוח פרויקט תוכנה במחשב ושימוש בעקרונות התכנות על מנת להגיע ליישום מסוג תוכנה. עיקר הפרויקט הינו פיתוחי, לימודי והתנסותי.

אופטיקה גיאומטרית הינה תחום במדעי הפיזיקה העוסק בתופעות של אור המתפשט במרחב בקווים ישרים. תחום זה מתבסס על שימוש מרובה בתחום הגיאומטריה על מנת להסביר תופעות של אור הנע בקו ישר. במהלך העבודה הרציפה על הפרויקט נעשה שימוש מרובה בהנחות, עקרונות, עצמים, חוקים ונוסחאות מתחום האופטיקה הגיאומטרית. הפרויקט ינסה לבטא מספר נושאים מרכזיים ובסיסיים בתחום זה.

בימנו תוכנות דימוי הינם מרכזיות בתחומים רבים במדע. לתחום האופטיקה גיאומטרית, כמו נושאים רבים במדע, ישנם תוכנות רבות ומגוונות המדמות אותה. תוכנות אלו נותנות מענה הן להצגה וויזואלית של התופעות והן חישובים שונים בתחום. הפרויקט לא מנסה להחליף ולשפר באופן מהותי את אותן תוכנות רבות, כי אם להוות אלטרנטיבה בסיסית, לימודית וידידותית למשתמש. מטרת הפרויקט היא לדמות את התחום באופן פשטני ונוח. חשוב לציין שהמוצר המוגמר לא מתיימר להוות כלי מדעי אמפירי ולא להיות מדויק ככזה.

# מטרת העבודה

המוצר הסופי מתוכנן להיות תוכנת מחשב בסביבת חלונות, אשר תשמש כלי לצורך הדמיית מערכת פיזיקלית של אופטיקה גיאומטרית. המוצר אמור לכלול ממשק משתמש נוח ושימושי, הדמיה גראפית של מערכת המבוססת על אופטיקה גיאומטרית והצגת נתונים הנוגעים למערכת. התוכנה תיתן למשתמש כלים רבים הנוגעים למערכת אופטית כגון מקורות אור, עדשות ומראות, כלים על מנת לשנות המאפיינים שלהם וכמו כן כלים על מנת להזיז ולשנות את המצב הנתון. המוצר ישמש כתוכנת סימולציה אינטואיטיבית באופן חיצוני ותפעל באופן מתמטי וחישובי "מאחורי הקלעים". המוצר הסופי לא יהיה כלי מדעי של ממש אך כן ישאף לספק הערכה חישובית ומתמטית קרובה בהתבסס על משוואות וחוקים מהתחום.

המוצר אמור להציג בצורה חזותית נושאים מרכזיים בתחום האופטיקה הגיאומטרית כגון: מקורות אור, עדשות מרכזות ומפזרות, מראות וחוסמי אור. התוכנה תחשב עפ"י נתונים ומתמטיקה המתאימים לתחום כגון: זוויות, מרחקים, גיאומטריה אנליטית וטריגונומטריה.

המערכת תספק מענה לשאלות בסיסיות מתחום האופטיקה הגיאומטרית ותספק תשובות משוערות, הן באופן חזותי והן באופן אינפורמטיבי. דוגמאות לשאלות שהמוצר יוכל לענות עליהן להלן: כיצד "יתנהג" האור בפגיעה בעדשה או במראה? כיצד המערכת תראה כאשר נציב מראות רבות? איכן תמצא הדמות ביחס לעדשה? וכיוצא בכך.

# ניסוח וניתוח של בעיות הפיתוח והבעיה האלגורתמית

**דרישה:** סביבת עבודה מתאימה לפיתוח תוכנת מחשב. **פירוט:** התוכנה אמורה להיות מפותחת בסביבת עבודה נוחה ואשר מאפשרת כלים ואפשרויות שונות על מנת לממש את מטרות התוכנה.

**דרישה:** ייצוג של עצמים שונים מתחום האופטיקה. **פירוט:** התוכנה אמורה לדמות כלים ועצמים שונים מתחום האופטיקה הגיאומטרית. נחוץ דרך לייצג כול עצם עם התכונות והיכולות הייחודיות לו.

**דרישה:** הצגה חזותית של מערכת אופטית. **פירוט:** התוכנה אמורה להציג באופן גראפי את המערכת האופטית ולעדכן אותה. נחוץ ייצוג גרפי לכול עצם וכן שיהיה לו: מיקום, צבע, גודל וכדומה. כמו כן על הגרפיקה להתעדכן במהירות בהתאם לשינויים במערכת.

**דרישה:** אלגוריתמים של עצמים שמשפעים על תנועת האור. **פירוט:** נחוץ אלגוריתמים של כול העצמים שמשנים את כיוון האור במערכת. עצמים שנדרש להם יישום אלגוריתמים אלו: עדשות, מראות, חוסם אור ותווך.

**דרישה:** אלגוריתמים לסיבוב נקודות דו ממדיות סביב נקודת ציר. **פירוט:** התוכנה אמורה לאפשר למשתמש לסובב את כול העצמים בזווית רצויה. נדרש אלגוריתם על מנת לסובב את העצם סביב נקודת המרכז שלו.

**דרישה:** אלגוריתמים לזיהוי של חיתוך בין אור (קו) עם עצם (שטח מסוים). **פירוט:** כאשר האור פוגע בעצם לדוגמא עדשה, התוכנה אמורה לזהות היכן בדיוק התקיימה הפגיעה.

# תיאור אלגוריתמים ושיטות קיימות לפיתרון בעיות הפיתוח

**דרישה:** סביבת עבודה מתאימה לפיתוח תוכנת מחשב. **אלגורתמים/שיטות קיימות הנותנות מענה על הדרישה:** א. סביבתVisual Studio (.Net) של חברת מיקרוסופט (C#,C++,VB)[[1]](#footnote-1) ב. סביבת NetBeans (Java)[[2]](#footnote-2) ג. סביבות עבודה צד שלישי (מגוון רחב של שפות) **למה לבחור בו:** א. סביבת עבודה אמינה ויש לה ניסיון בה. כמו כן למדתי תכנות בשפת #C . ב. סביבת עבודה שנותנת פיתוח חוצה פלטפורמות בנוסף יש לי ניסיון מסוים בשפה Java. ג. חוצה פלטפורמות ומאפשר עבודה במגוון שפות תכנות.

**דרישה:** ייצוג של עצמים שונים מתחום האופטיקה. **אלגורתמים/שיטות קיימות הנותנות מענה על הדרישה:** ייצוג של עצמים ע"י מחלקות ושימוש בתחום 'תיכונות מונחה עצמים'. **למה לבחור בו:** התחום 'תכנות מונחה עצמים' הינו תחום מאוד מבוסס ומרכזי בתכנות מודרני ועונה בדיוק על הצרכים בייצוג עצמים בפרויקט זה. בנוסף יש לי ניסיון אישי אתו.

**דרישה:** הצגה חזותית של מערכת אופטית. **אלגורתמים/שיטות קיימות הנותנות מענה על הדרישה:** א. ייצוג עצמים בצורה גראפית ע"י ספריות צד שלישי (לדוגמא: OpenGl ) [[3]](#footnote-3) ב. ייצוג עצמים בצורה גראפית ע"י ספרית הגרפיקה של שפת התכנות C# (.Net) [[4]](#footnote-4) **למה לבחור בו:** א. ספריות אלו ובפרט OpenGl הם ספריות גרפיקה מאוד עשירות ומתקדמות. ב. ספריה יחסית פשוטה ונוחה לשימוש.

**דרישה:** אלגוריתמים לסיבוב נקודות דו ממדיות סביב נקודת ציר.[[5]](#footnote-5) **אלגורתמים/שיטות קיימות הנותנות מענה על הדרישה:** אלגוריתם חישובי מתמטי המבוסס על טריגונומטריה. **למה לבחור בו:** עונה על הצורך האלגוריתמי.

# 

# 

# הפתרונות המוצעים

**דרישה:** סביבת עבודה מתאימה לפיתוח תוכנת מחשב. **הפיתרון או האלגוריתם הנבחר:** סביבת Visual Studio (.Net) של חברת מיקרוסופט (C#,C++,VB) **נימוק הבחירה בו:** לא ראית צורך בפיתוח תוכנה חוצה פלטפורמות כחלק ממטרות הפרויקט. יש לי ניסיון אישי עם סביבה זאת והיא נוחה ואמינה. בנוסף אני מעונין לפתח בשפת C# שאני מרגיש בטוח אתה. אני גם לא מרגיש בנוח עם שפת Java ולכן גם לא בחרתי ב NetBeans. **הערות:** בסביבה זאת אני יפתח יישום מסוג Widows Forms שמתאימה לכול הצרכים של הפרויקט.

**דרישה:** ייצוג של עצמים שונים מתחום האופטיקה. **הפיתרון או האלגוריתם הנבחר:** ייצוג של עצמים ע"י מחלקות ושימוש בתחום 'תיכונות מונחה עצמים'. **נימוק הבחירה בו:** 'תכנות מונחה עצמים' הוא תחום שאני מרגיש בו נוח ומאוד חשוב בתכנות. השימוש בו מאוד מתאים והולם לצרכים של הפרויקט. **הערות:** כול המחלקות תיהנה מפורטות בצורה מלאה בחלק 'תיאור המודולים של מערכת התוכנה' בהמשך.

**דרישה:** הצגה חזותית של מערכת אופטית. **הפיתרון או האלגוריתם הנבחר:** ייצוג עצמים בצורה גראפית ע"י ספרית הגרפיקה של שפת התכנות C#. **נימוק הבחירה בו:** אין צורך בפרויקט בגרפיקה תלת ממדית או מתקדמת כלל. ויותר מכך ספרית הגרפיקה של שפה זו היא נוחה ומוכרת לי. ספריה זאת עונה על כול הצרכים של הפרויקט ואין צורך בספריה יותר מורכבת ולכן בחרתי בה. **הערות:**

**דרישה:** אלגוריתמים של עצמים שמשפעים על תנועת האור. **הפיתרון או האלגוריתם הנבחר:** אלגוריתמים מקוריים שלי לכול עצם על בסיס נוסחאות מהתחום האופטיקה הגיאומטרית. **נימוק הבחירה בו:** אלגוריתמים אלו הם מאוד מרכזיים ולכן ראיתי לנכונה לעבוד עליהם בעצמי. כמו כן לא נראה לי שקיימים חלופות פומביות מוכרות. **הערות:** האלגוריתמים לא מושלמים ויעלים באופן מיוחד אך נותנים מענה למרבית המצבים/דרישות.

**דרישה:** אלגוריתמים לסיבוב נקודות דו ממדיות סביב נקודת ציר. **פיתרון מוצע:** אלגוריתם חישובי מתמטי המבוסס על טריגונומטריה. **נימוק הבחירה בו:** עונה באופן מלא על הצורך האלגוריתמי. **הערות:**

**דרישה:** אלגוריתמים לזיהוי של חיתוך בין אור (קו) עם עצם (שטח מסוים). **פיתרון מוצע:** שימוש בספרייה הגרפיקה של שפת התכנות C# ושימוש בכלים בספריה כדי לבדוק את שטח החיתוך בין שטח מרובע (המייצג את האור) ושטח המייצג את העצם. אם שטח החיתוך ריק אין חיתוך בין השניים ואם לא ניתן להשתמש בשטח החיתוך לצורך חישובים. **נימוק הבחירה בו:** זה הפתרון הכי הולם שמצאתי והוא אכן נותן מענה מספיק טוב לדרישה. **הערות:**

# תיאור המודולים של מערכת התוכנה

הפרויקט פותח באמצעות שימוש בתחום 'תכנות מונחה עצמים'. תחום זה מרכזי בחשיבותו בתכנות בימנו ומתבסס על הפרדה של הקוד לעצמים שונים. התוכנה מורכבת ממספר רכיבים: מחלקות (עצמים) , ממשקים, מחלקות שירות ומחלקות גראפיות/חלונות. להלן פירוט של המודלים המרכזיים שנעשה בהם שימוש, מאפייניהם ותפקידם:

מחלקות (עצמים):

שם: PhysicalObject (עצם פיזיקלי) מאפיינים: מחלקה מסוג אבסטרקט (מופשטת) תפקיד: מייצגת עצם מופשט בעל מאפיינים פיזיקליים כגון מיקום, צבע וזוויות. מחלקה זאת משמשת כמחלקת 'אב' לכול שאר העצמים האופטיים (עדשה, מראה, מקור אור וכ"ו).

שם: Light (קרן אור) מאפיינים: מחלקה המתארת עצם שניתן לתת לו מופע (instance) תפקיד: המחלקה מייצגת קרן אור של קו ישר. המחלקה בבסיסה מיוצגת על ידי שתי נקודות המגדירות קו ישר וצבע האור.

שם: Lens (עדשה) מאפיינים: מחלקה המתארת עצם שניתן לתת לו מופע (instance) תפקיד: המחלקה מייצגת עדשה אופטית. המחלקה מכילה מאפיינים הן של עצם פיזיקלי והן מאפיינים רבים ומגוונים של כול עדשה כגון: רוחק מוקד, רדיוסים המייצגים את העדשה ועוד. פעולות מרכזיות: פונקציה שבודקת האם אור פוגע בעדשה, פונקציה שמשתנה את הכיוון של רשימה של קרני אור (אינטראקציה של האור עם העדשה) ופונקציה שצובעת את העדשה בהתאם לרדיוסים שמגדירים אותה.

שם: Source (מקור אור) מאפיינים: מחלקה מסוג אבסטרקט (מופשטת) תפקיד: מייצגת עצם מסוג מקור אור מופשט. יש לה מאפיינים כללים כמו נקודת יציאת האור ומספר קרני אור שיוצאים מהמקור. מחלקה זאת משמשת כמחלקת 'אב' לכול עצם מסוג מקור אור ספציפי (מקור אור שבסיסו קו, מקור אור שבסיסו עיגול). פעולות מרכזיות: פונקציה שמציירת את העדשה ופונקציה שמציירת את הקרן אור שיוצאת מהמקור.

שם: StraightSource (מקור אור שבסיסו ישר) מאפיינים: מחלקה המתארת עצם שניתן לתת לו מופע (instance) תפקיד: המחלקה מייצגת מקור אור שבסיסו קו ישר. בנוסף למחלקת האב שלה יש לה גם פעולה שנותנת את נקודת המוצא של קרני האור.

שם: CircularSource (מקור אור שבסיסו עיגול) מאפיינים: מחלקה המתארת עצם שניתן לתת לו מופע (instance) תפקיד: המחלקה מייצגת קרן אור של קו ישר. המחלקה בבסיסה מיוצגת על ידי שתי נקודות המגדירות קו ישר וצבע האור.

שם: Block (חוסם אור) מאפיינים: מחלקה המתארת עצם שניתן לתת לו מופע (instance) תפקיד: המחלקה מייצגת גוף ריבועי שגורם לכך שכול קרן אור שפוגעת בו עוצרת בו. פעולות מרכזיות: פעולה הבודקת האם קרן אור מסוימת פוגעת בו ופעולה שגורמת לקרן אור הפוגעת בו לשנות את מסלולה ולעצר בו.

שם: Medium (תווך) מאפיינים: מחלקה המתארת עצם שניתן לתת לו מופע (instance) תפקיד: המחלקה מייצגת חצי עיגול בעל מקדם שבירה הנתון לשינוי. פעולות מרכזיות: פעולה הבודקת האם קרן אור מסוימת פוגעת בו ופעולה שגורמת לקרן אור הפוגעת בו לשנות את מסלולה בהתאם לחוקי השבירה (סנל).

שם: Mirror (מראה) מאפיינים: מחלקה המתארת עצם שניתן לתת לו מופע (instance) תפקיד: המחלקה מייצגת ריבוע שהוא מראה. פעולות מרכזיות: פעולה הבודקת האם קרן אור מסוימת פוגעת בו ופעולה שגורמת לקרן אור הפוגעת בו לשנות את מסלולה בהתאם לחוקי החזרת האור.

ממשקים:

שם: IDrawable (בר צביעה) מאפיינים: ממשק הדורש מימוש פעולות תפקיד: הממשק מגדיר פעולות שנדרשות למימוש עיקרון הפולימורפיזם על מנת שכול עצם המוגדר כ'בר צביעה' יצבע באופן שהוא מגדיר.

שם: IInteract (בר אינטראקציה עם אור) מאפיינים: ממשק הדורש מימוש פעולות תפקיד: הממשק מגדיר פעולות שנדרשות למימוש עיקרון הפולימורפיזם על מנת שכול עצם המוגדר אינטראקציה' יוכל לבצע אינטראקציה עם אור.

שם: ISource (בר שימוש כמקור אור) מאפיינים: ממשק הדורש מימוש פעולות. תפקיד: הממשק מגדיר פעולות שנדרשות לכול עצם המוגדר כמקור אור.

מחלקות שירות/עזר:

שם: DrawingHelper (עזר בצביעה) מאפיינים: מחלקה שירות המכילה פעולות סטטיות (פעולות מחלקה). תפקיד: המחלקה מאפשרת גישה למספר פעולות העוזרות בצביעה וגרפיקה.

שם: AlgoritemHelper (עצם פיזיקלי) מאפיינים: מחלקה שירות המכילה פעולות סטטיות (פעולות מחלקה). תפקיד: המחלקה מאפשרת גישה למספר פעולות אשר מבצעות את האלגוריתמים הראשיים.

שם: MathHelper (עצם פיזיקלי) מאפיינים: מחלקה שירות המכילה פעולות סטטיות (פעולות מחלקה). תפקיד: המחלקה מאפשרת גישה למספר פעולות העוזרות בצביעה וגרפיקה.

## Windows Forms (מחלקה גרפית/חלונות)

שם: Main\_Form (חלון ראשי) מאפיינים: מסוג windows forms תפקיד: החלון הראשי המכיל את הרוב המוחלט של הממשק משתמש.

שם: About(חלון אודות) מאפיינים: מסוג windows forms תפקיד: חלון משני המציג מידע אודות התוכנה.

שם: AdvanceOptions ( חלון אפשרויות מתקדמות) מאפיינים: מסוג windows forms . תפקיד: חלון משני המציג אפשרויות מתקדמות.

שם: Feedback (חלון משוב) מאפיינים: מסוג windows forms תפקיד: חלון משני המציג אפשרויות לשליחת משוב על התוכנה.

## מבני נתונים שנעשה בהם שימוש:

שם: רשימה סיבת הבחירה בו: הצגה של מספר דינמי של עצמים תפקידים עיקריים: הצגה של כול העצמים האופטיים שנעשה עליהם הדמיה. בנוסף לכך שימוש ברשימה של עצמים המייצגים קרן אור כך נוצר המסלול שעובר קרן אור מתחילתו לסופו.

שם: מחסנית סיבת הבחירה בו: לאחסן משתנים "אחד על גבי השני" תפקיד עיקרי: שמירה של הרשימה המכילה את מרבית המידע אודות המערכת ברגעים שונים וכך כאשר נוציא מראש המחסנית את הרשימות נוכל להחזיר אותם כפי שהם היו בעבר. למעשה המחסנית מאפשרת לעשות פעולת חזור 'undo'.

שם: תור סיבת הבחירה בו: לאחסן משתנים ברצף כרונולוגי (מהראשון שהוכנס לאחרון) תפקיד עיקרי: שמירה של איברים שהוצאו על ידי פעולת חזור 'undo' על מנת להחזיר את המידע כפי שהיה לפני פעולת החזור. כלומר התור מאפשר לעשות פעולת 'redo'.

להלן תרשים זרימה כללי (סכמטי) של המחלקות המרכזיות בתוכנה:

ISource

ממשק בר שימוש כמקור

IDrawable

ממשק בר צביעה

IInteract

ממשק בר אינטראקציה

PhysicalObject

עצם פיזיקאלי מופשט

Source

מקור אור מופשט

Mirror

מראה

Lens

עדשה

Medium

תווך

Block

חוסם אור

CircularSource

מקור אור משטח עגול

StraightSource

מקור אור משטח ישר

Light

קרן אור

AlgoritemHelper

פונקציות אלגוריתמיות

DrawingHelper

פונקציות צביעה

MathHelper

פונקציות חישוב

AdvanceOptions

חלון אפשרויות מתקדמות

Feedback

חלון משוב

Main\_Form

חלון ראשי

About

חלון אודות

מקרא:

מחלקת חלונות

WinForms

מחלקת שירות

מחלקת אובייקט

מחלקת ממשק

מחלקה אבסטרקטית

מימוש ממשק (realization) הורשה (Inheritance)

# סביבת עבודה

שפת תכנות: C# (סי שארפ)

סביבת עבודה: Microsoft Visual Studio Ultimate 2013

סוג תוכנה: Windows Forms Application

כלים: Adobe Photoshop CC 2015

מערכות/ספריות: Microsoft .Net Framework

# הערכת פיתרון

## השוואה בין התוכנה לבין ישום קיים

לצורך השוואה אקח אתר המדמה דמות של עצם הנוצר ע"י עדשה. ההשוואה תתבצע רק בנושא של עדשות בתוכנה. תוכנת דימוי זאת[[6]](#footnote-6) היא חלק מאתר בשם 'phet.colorado.edu' שבו תוכנות דימוי רבות. אישית אני חושב שזאת תוכנת דימוי מאוד טובה, מקצועית וידידותית למשתמש. אמנע מספר נקודות דמיון ושוני בין תוכנה זו לבין זו שלי.

נקודות דימיון:

- דמות "אמיתית" ודמות "מדומה" של מקור אור. - הזזה של מקור האור באמצעות העכבר. - הדמיה של עדשה מרכזת. - שינוי ערכי רדיוס של העדשה ואורכה.

יתרונות של האתר על פני התוכנה: האתר מאפשר להראות דמות של מקור אור לא נקודתי, שינוי אינדקס (מקדם שבירה) של עדשה, שימוש ב3 קווים מנחים כדי להראות את הדמות, שימוש במסך (שהאור עובר אותו), שימוש מאוד פשוט ונוח וגישה נוחה דרך אתר אינטרנט.

יתרונות של התוכנה על פני האתר:עדשה מפזרת, שינוי ערכי שני הרדיוסים של העדשה וכן ייצוג עדשה על ידי רדיוסים שליליים, הצגת הדמות לא רק של ע"י קווים מנחים אלה גם את השינוי שנגרם לקרן אור מסוימת, הצגת נתונים מספריים רבים יותר (מרחק מעדשה, מרחק), שינוי צבעים וריבועי של עדשות ומקורות אור.

## הבדלים בין התוכנה במצבה הנוכחי לתוכנה המתוכננת

דברים שהיו בתוכנה שתוכננה וחסרים בתוכנה במצבה הנוכחי:

- חישובים גם לעדשות שהן לא עדשות דקות ובהן יש התייחסות לעובי העדשה.

- חישובים על מקדם השבירה של העדשה.

- שימוש מאוד פשוט ונוח (כמו באתר עמו השוואתי הנ"ל).

- אלגוריתמים שעובדים בכול מצב.

דברים שהצלחתי להוסיף לתוכנה במצבה הנוכחי שלא תוכנן במקור:

- עבדתי קצת על עצם שהוא תווך (ריבוע שיש לו מקדם שבירה בר שינוי), האלגוריתם עדיין לא מושלם אבל בעיקרון עצם זה מדגים את חוק סנל (הנוגע לשינוי בכיוון האור כתוצאה משינוי תווכים). עצם זה לדוגמא פועל כמו במראה אם הזווית הפגיעה גדולה מזוויות הקריטית אם יש כזו (על פי חוק סנל), ומציגה קרן אור הנעה ממקדם שבירה קטן לגדול ולהפך.

- הצלחתי לעבוד על כול מיני דברים קטנים הקשורים לממשק המשתמש (שינוי צבעים, חלון אודות, חלון משוב, שמירה של המערכת, הדפסה ועוד הרבה דברים קטנים).

## המלצות לשיפור התוכנה במצבה הנוכחי

התוכנה במצבה הנוכחי לא במצב של מוצר מוגמר ולא מתיימר להיות. לצערי לא הספקתי לעשות את כול מה שרציתי. לעומת זאת אני מאמין שהספקתי לא מעט ומרגיש שהצלחתי הרבה דברים אחרים.

שיפורים אפשריים:

- שיפור משמעותי באלגוריתמים כך שיעבדו בכול מצב.

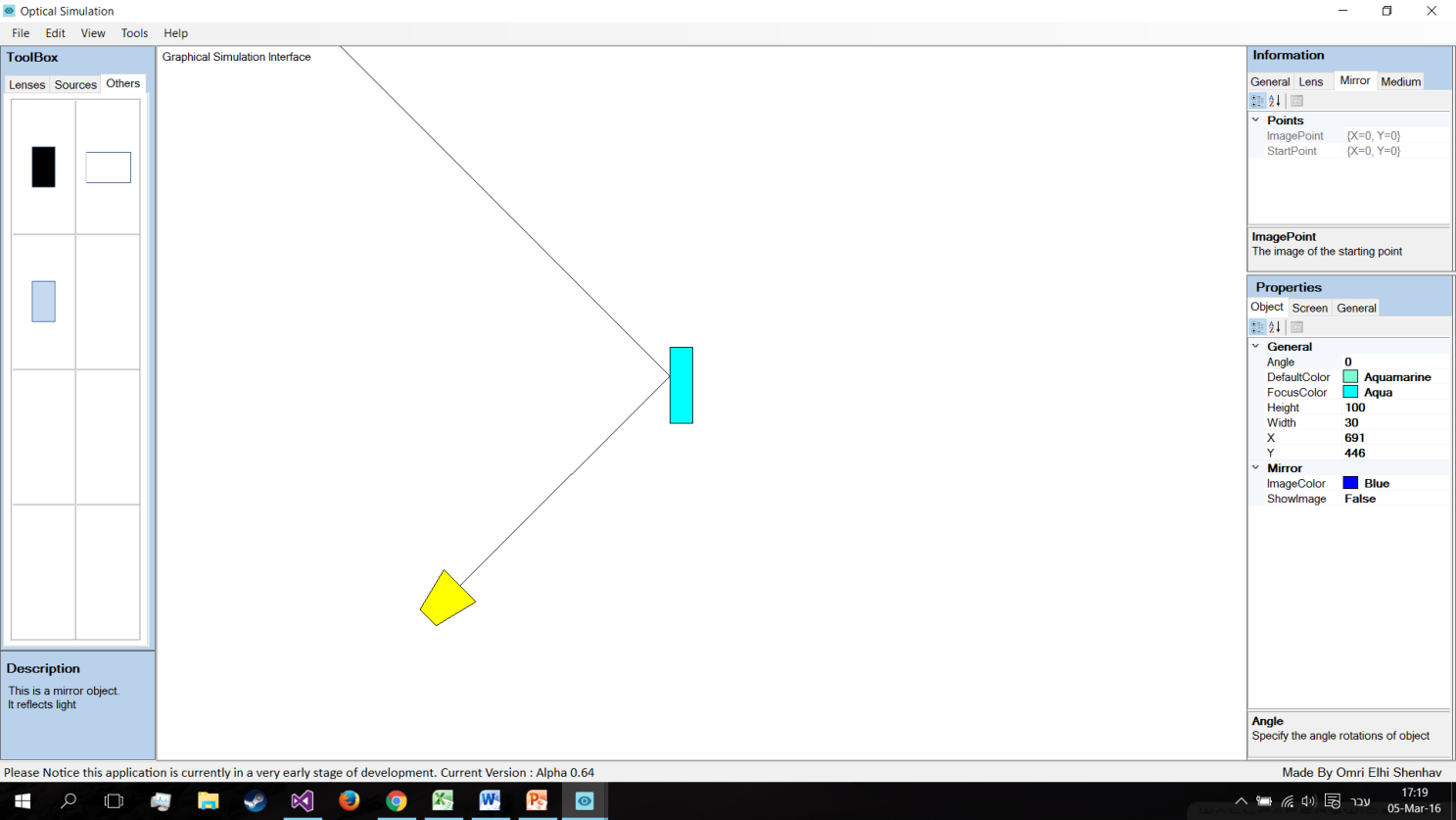
- אלגוריתם של עדשות שאינם בקירוב עדשות דקות. כלומר אלגוריתם של עדשות כפי שהם במציאות.

- שיפור משמעותי לאופן הפעלת התוכנה כך שתהיה הרבה יותר אינטואיטיבית וקלה.

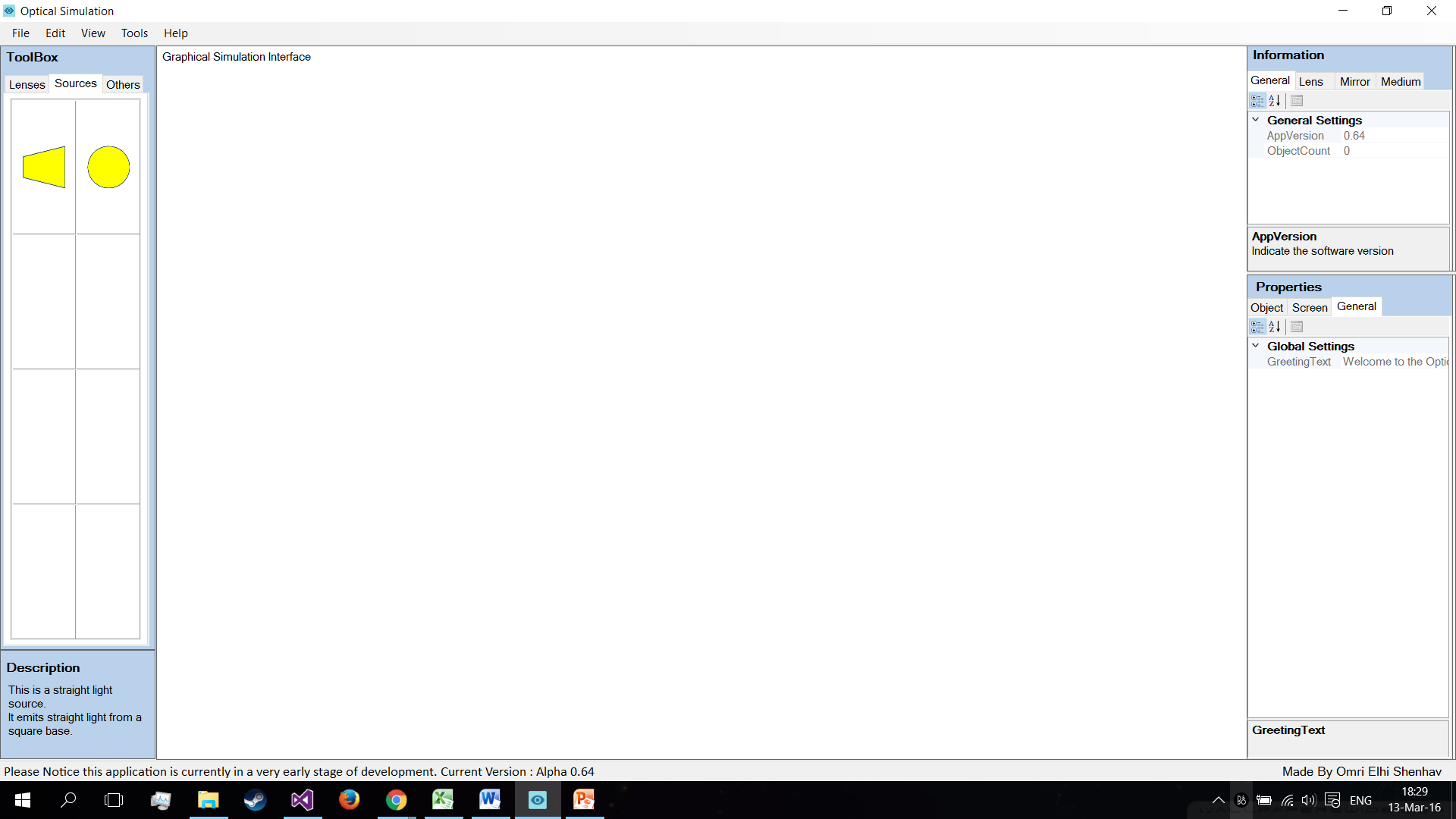
# תיאור הממשק למשתמש

התוכנה כוללת ממשק משתמש בעל רכיבים רבים ושונים, הסביר באופן מפורט על הרכיבים השונים והסביר בפירוט על תפקידים וכן אדגים אותם מילולית וויזואלית.

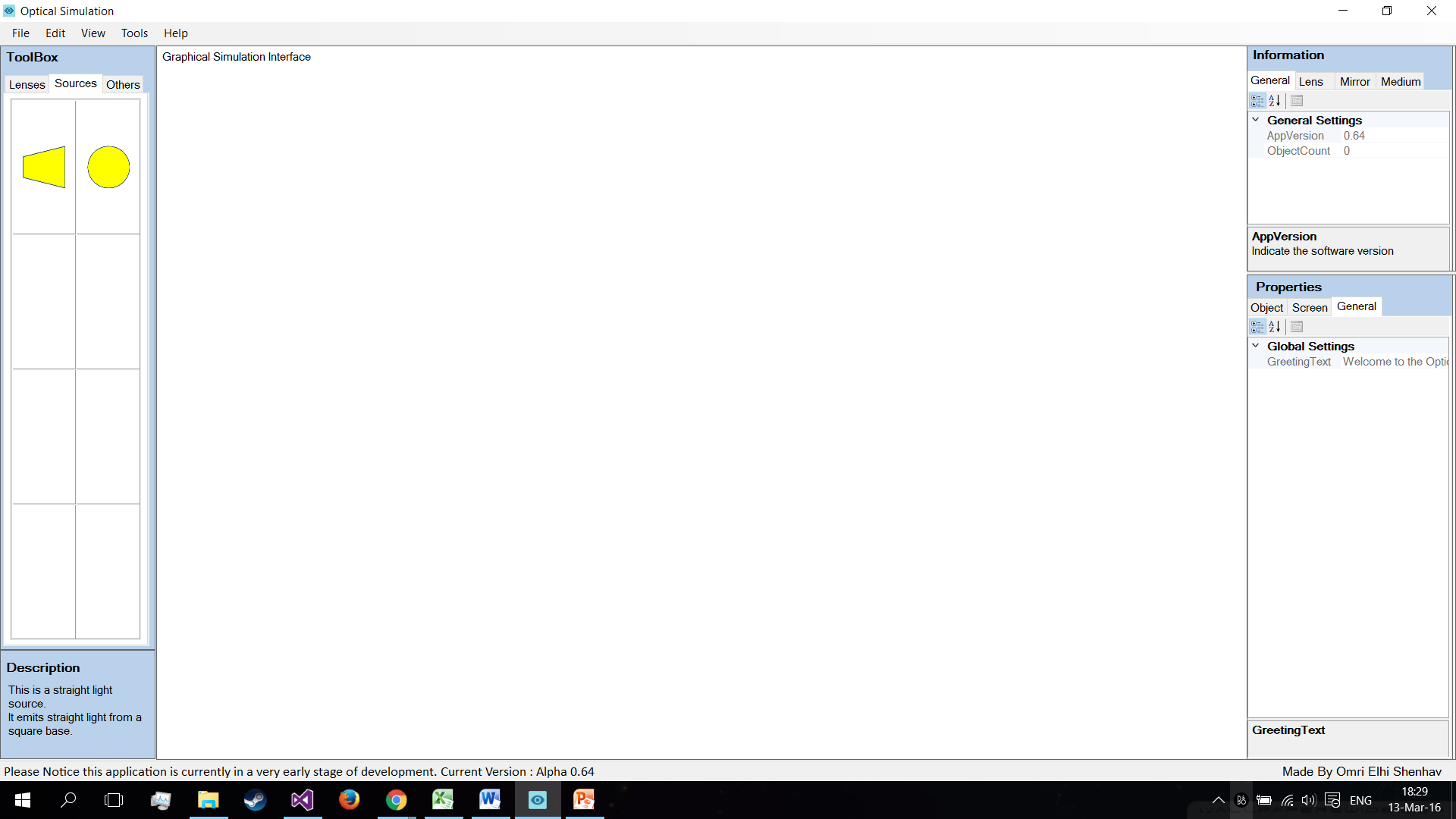
שם: “Screen” (מסך/תצוגה גרפית) תפקיד: ריבוע גדול במרכז הממשק הראשי. על פני הרכיב הזה יופיע ההדמיה הגרפית. על רכיב זה "מצוירים" כול העצמים במערכת שמדמה אופטיקה גיאומטרית. המשתמש יכול ללחוץ על עצמים שמצוירים על המסך ולהזיז אותם. כאשר "בטעות" הוא מזיז את העצם כך שמרכזו מחוץ למסך, העצם ימחק. בנוסף מספר העצמים על המסך בבת אחת מוגבל. דוגמאות: על המסך יופיע ריבוע שהוא מראה, וקו (המייצג קרן אור) שמגיע אליו ולאחר מכן סוטה ממסלולו לכיוון אחר. תמונה:

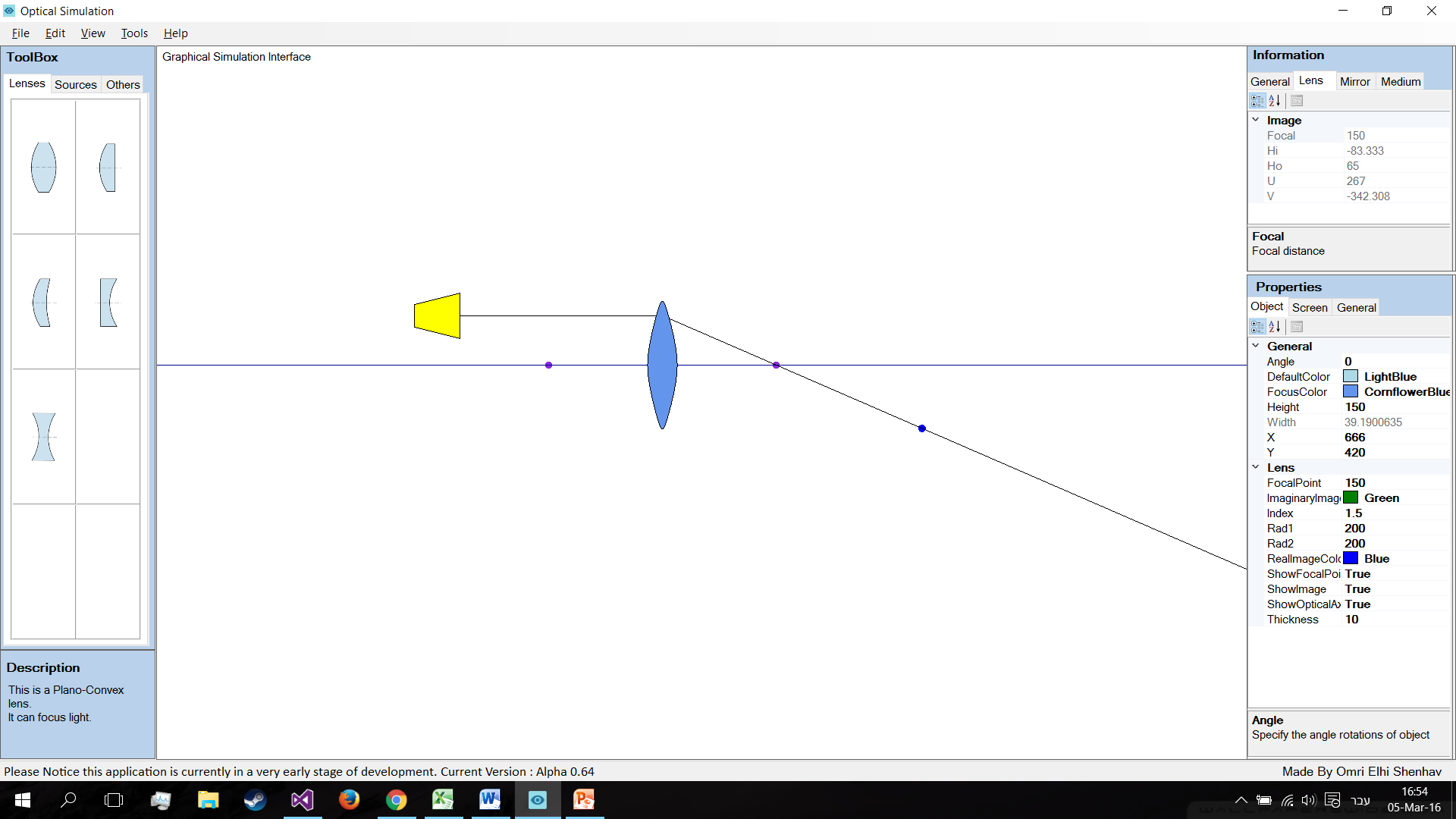


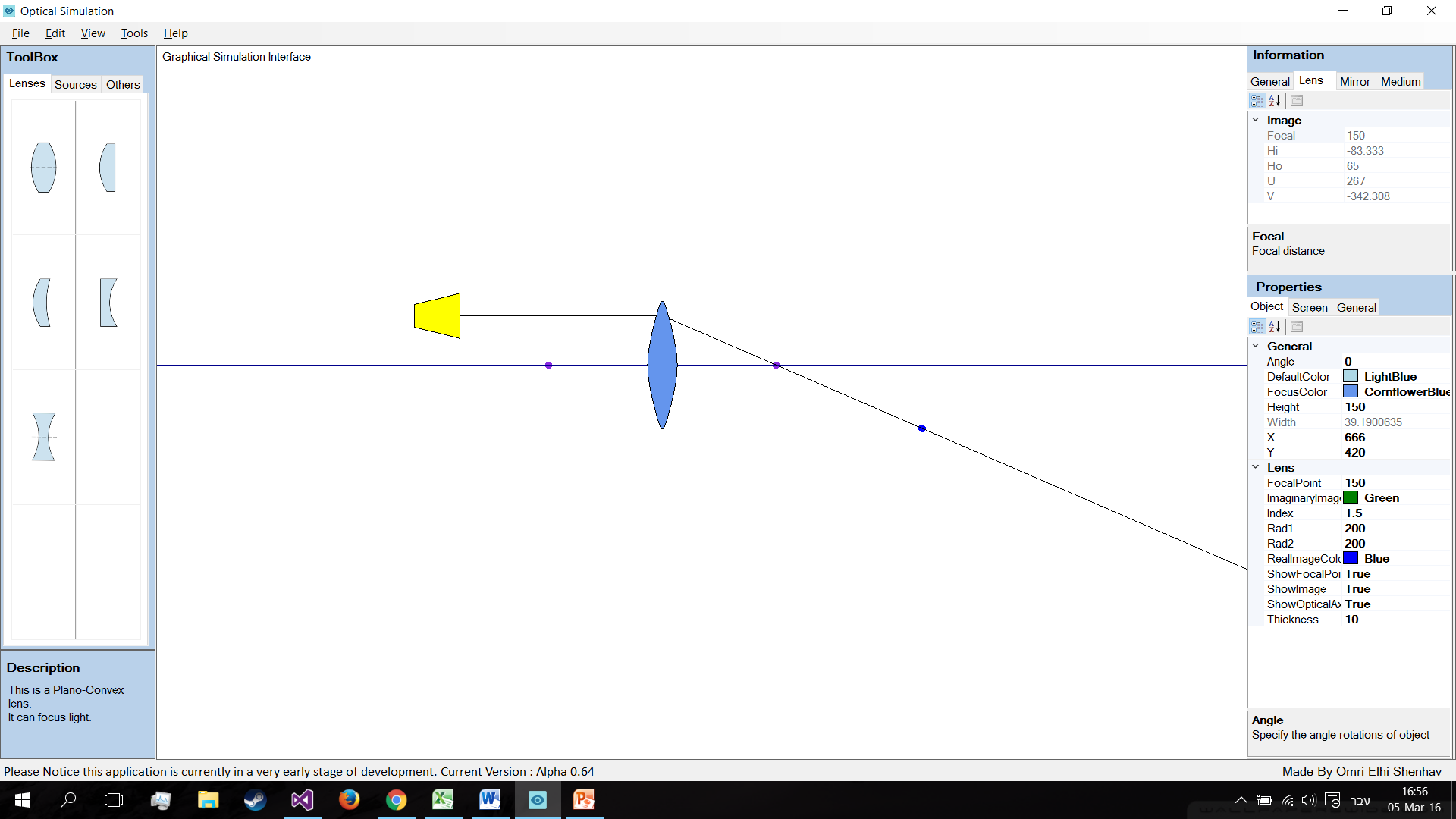
שם: “ToolBox” (ארגז כלים) תפקיד: רכיב זה נמצא בממשק הראשי ומציג בפני המשתמש תמונות של הכלים שהוא יכול להוסיף למערכת ההדמיה. המשתמש לוחץ פעם אחת על תמונה של הכלי הרצוי ופעם אחת על המסך, והכלי הרצוי יופיע על המסך. רכיב זה מחולק לנושאים ונגיש למשתמש. דוגמאות: בתוך ארגז הכלים יש תמונה של עיגול שהוא מקור אור. ניתן ללחוץ על התמונה ולאחר מכן במרכז המסך. התוכנה תציג במרכז המסך ציור של מקור האור בצורת העיגול.

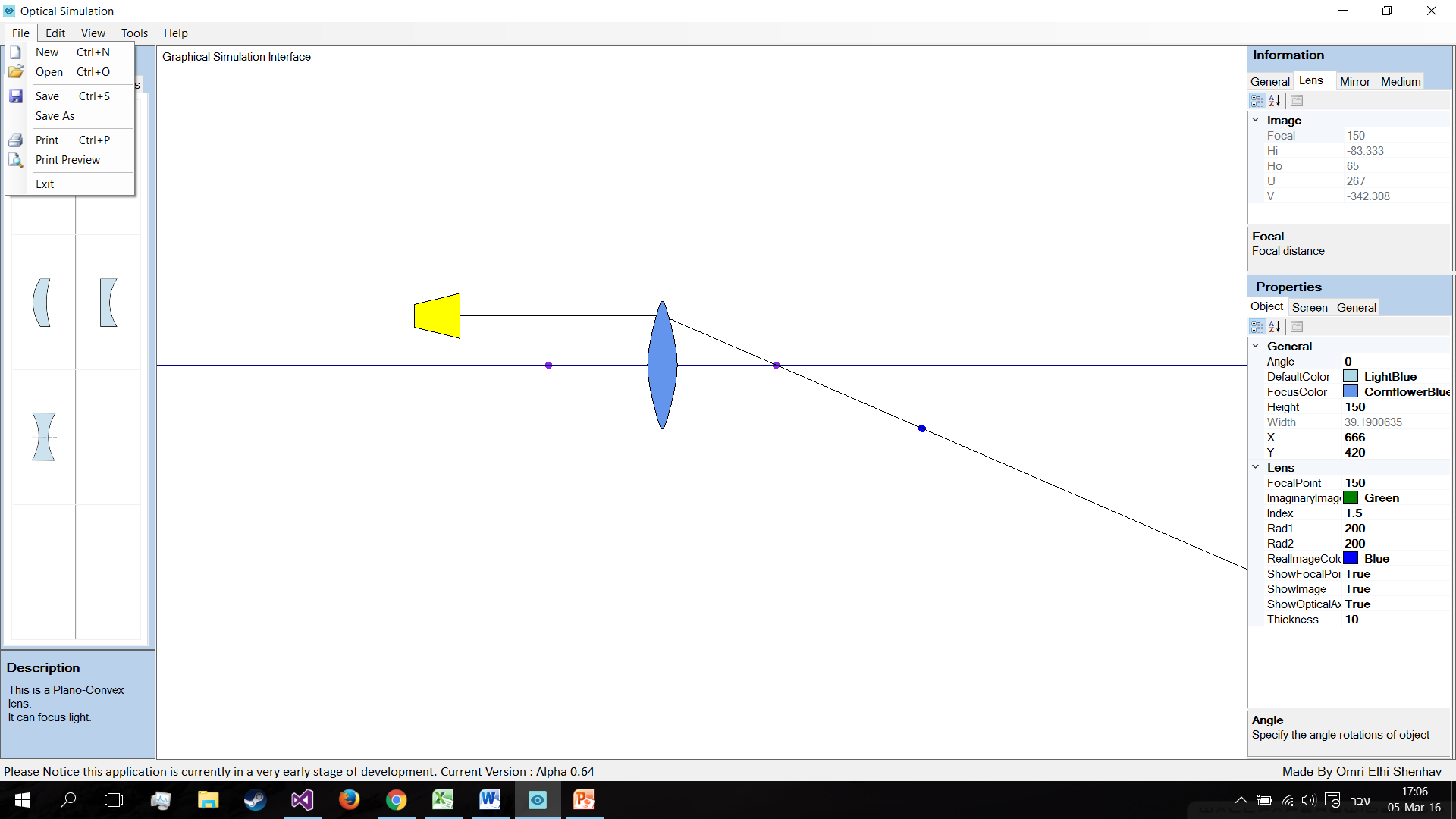
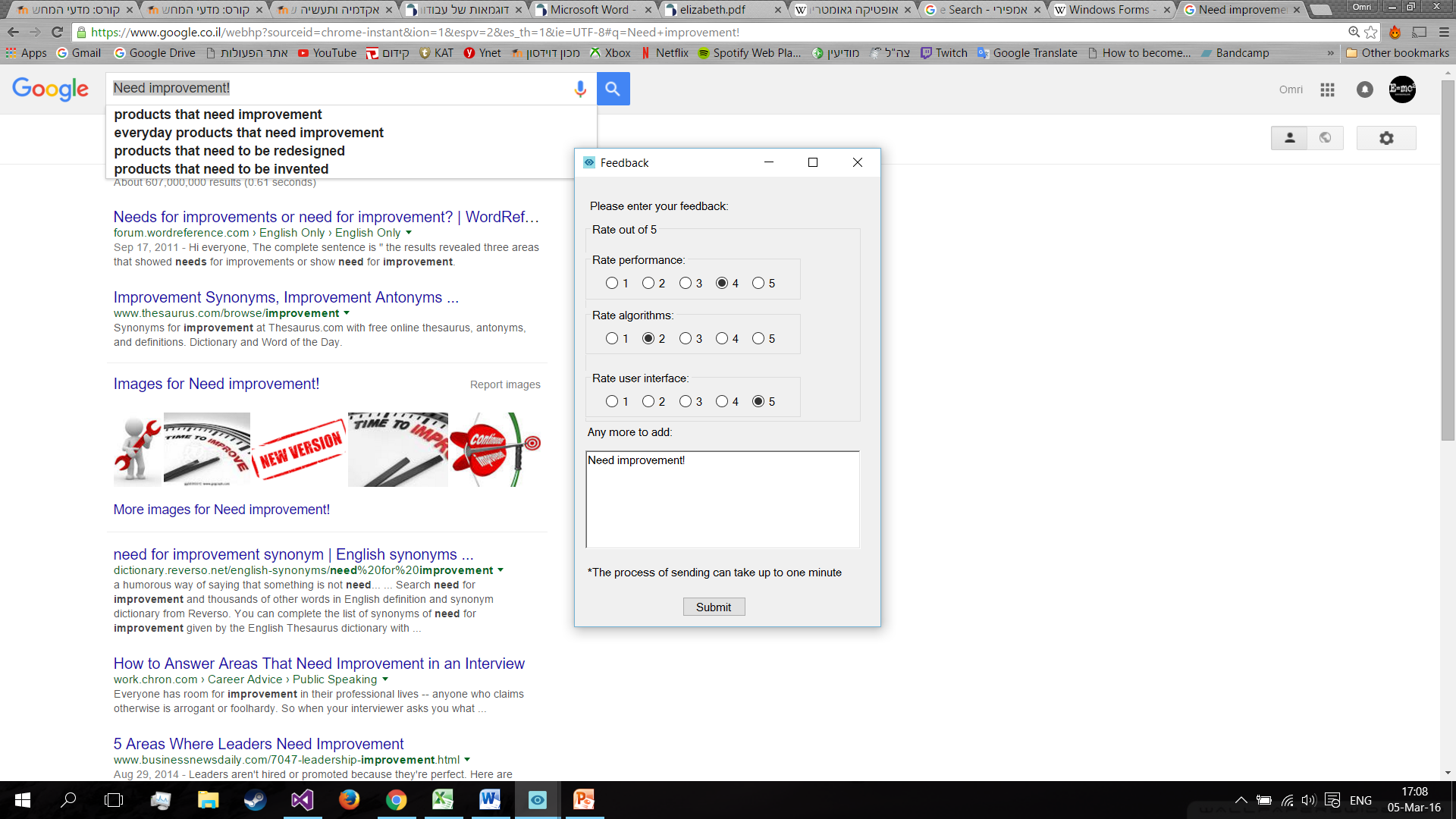
תמונה:

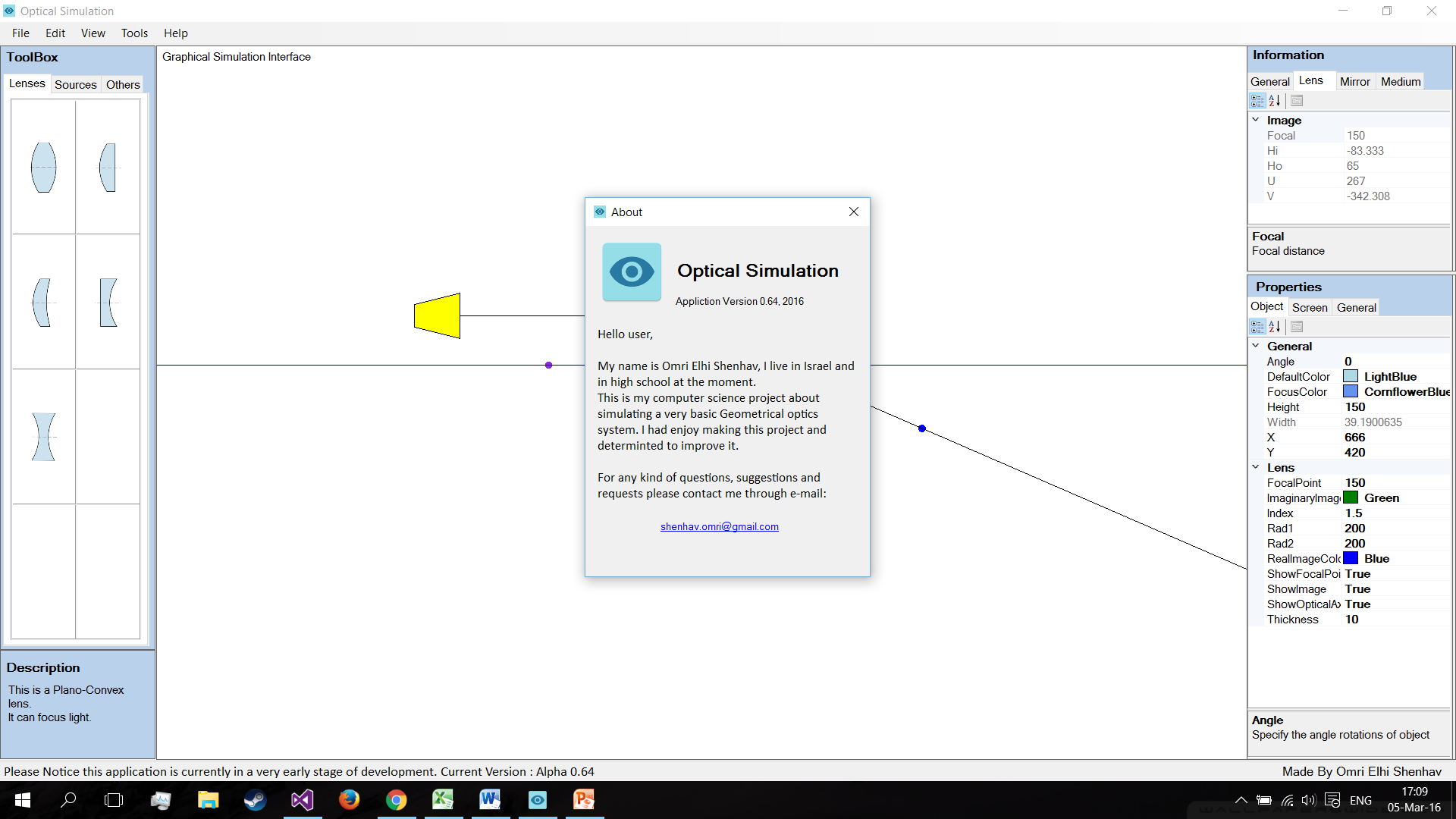
שם: “Description” (תיאור מילולי) תפקיד: רכיב זה הינו ריבוע קטן בממשק הראשי המכיל טקסט משתנה. כאשר המשתמש מעביר את העכבר מעל כלים בארגז הכלים וחלקים אחרים בממשק הראשי, הטקסט בריבוע זה ישתנה ויתאר בפני המשתמש בצורה פשוטה על מה הוא "מרחף". דוגמאות: כאשר המשתמש מעביר את העכבר מעל התמונה של מקור אור, על רכיב זה יהיה כתוב הסבר שזהו מקור אור.

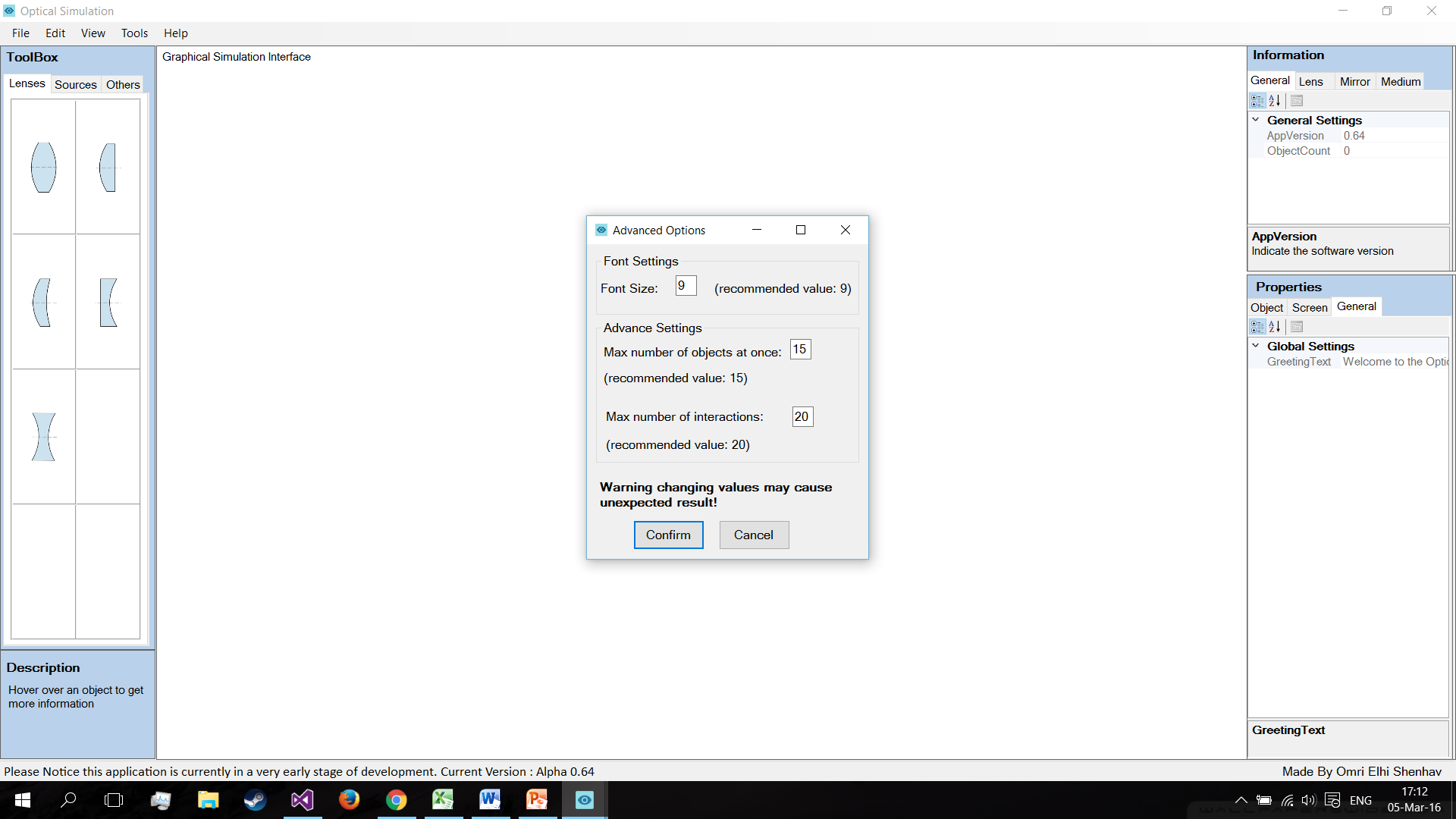
תמונה:

שם: “Information” (מידע) תפקיד: רכיב זה מציג מידע בסיסי על המערכת ומחולק לנושאים לפי סוג המידע. המידע יוצג כאשר המשתמש ילחץ על העצם שממנו הוא רוצה לקבל מידע. דוגמאות: כאשר המשתמש ילחץ על עדשה שפוגעת בה קרן אור יופיע מידע המתאר את המרחק של המקור מעדשה, רוחק המוקד של העדשה וכדומה. תמונה:

שם: “Properties” (מאפיינים) תפקיד: רכיב זה מציג מאפיינים של המסך ומאפיינים של העצם. רכיב זה מאוד משמעותי ומאפשר למשתמש לראות ולשנות מגוון רחב מאוד של ערכים. מאפייני המסך קבועים אך מאפייני העצם הם בהתאם לעצם הנבחר. עצם נבחר על ידי לחיצה עליו על המסך. במאפייני העצם יופיע כול המאפיינים המתאימים לעצם הנבחר, מידע רחב ומופרט עליו וכן אפשרות לשנות את ערכיו השונים. דוגמאות: במאפייני המסך ניתן לשנות את מהירות העיבוד הגרפי (כלומר כמה פעמיים המערכת מציירת את עצמה בשנייה) ואת צבע המסך. כאשר בוחרים בעדשה ניתן לשנות את רוחק המוקד שלה, את צבעיה ועוד. תמונה:

שם: רצועת תפריטים תפקיד: רכיב זה הוא שורה של תפריטים הזמינים למשתמש. להלן פירוט של האפשרויות שהתפריטים נותנים למשתמש: פרויקט חדש, פתיחת פרויקט, שמירת פרויקט בשם, הדפסה, תצוגה מקדימה של הדפסה, מחיקה העתקה הדבקה וגזירה של עצמים (עפ"י עצם שנבחר על לחיצה), מסך מלא, החלפת "ערכת נושא" (צבעים), אפשרויות מתקדמות, אפשרות לשלוח משוב (feedback) ואודות התוכנה. אפשרויות מתקדמות, אודות ומשוב הן אפשרויות שפותחות חלון חדש אשר ממלא את מטרתם. דוגמא: לחיצה על "help" ואז על "about" תפתח חלון חדש שיציג מידע אודות התוכנה. תמונות:





## C:\Users\Omri\Desktop\תמונה1.pngתמונה מייצגת של ממשק המשתמש הראשי

# רפלקציה

התהליך של העבודה התחיל במורה המלווה שלי שהציג בפני הכיתה את התוכנית של מכון דוידסון 'מדעי המחשב אקדמיה ותעשיה'. במהלך התוכנית נחשפתי לשלל אנשים ונושאים בתחומי מדעי המחשב שהרחיבו את ידיעותיי ונתנו לי השראה. לאחר שהתקבלתי לשלב השני של התוכנית, בו עושים פרויקט פיתוחי, התחלתי במהרה לעבוד על הפרויקט. נפגשתי עם המנחה והוא שכיוון אותי ועזר לי להתחיל. במשך השנה המנחה עזר לי רבות עם בעיות בפרויקט ועבדתי על הפרויקט כשבמקביל התמודדתי עם עומס לימודי רב. נתקלתי בכול מיני קשיים בדרך וכתבתי חלקים מהקוד מחדש כמה פעמים. לקראת הסוף השקעתי הרבה מאמץ לעבוד על הפרטים הקטנים כדי שתוצר יראה יותר שלם.

במבט לאחור העבודה על הפרויקט הייתה בדיוק מה שציפיתי והכינו אותי אליו ויותר מכך: מאתגרת, מעניינת, אישית ומשמעותית. יותר מהכול העבודה דרשה ממני הרבה, הן במישור האישי כמו שעות עבודה, משמעת ואחריות והן במישור המקצועי כמו התמודדות ולמידה עצמית. הוסיף לציין שזו הייתה חוויה שעזרה לי להבין באופן טוב כיצד לנהל פרויקט, דבר הכרחי ומועיל מאוד בעתיד המקצועי שלי ובכלל בחיים. חשוב לי להוסיף שאני מרגיש שאופי הפרויקט אפשר לי לבטא את עצמי ואת המשיכה שלי לנושאים הנלמדים ועל כן הדבר עזר לי מאוד לקדם אותו.

לתהליך הפיתוח, כמו כול תהליך, היו גם חלקים קשים ומאתגרים וגם חלקים מספקים ומהנים, ובסה"כ אני יוצא עם זיכרון חיובי מהחוויה כולה. אני מרוצה מהתוצאה של תהליך הפיתוח ואף שמחתי להציג אותה בפני אחרים. אני שמח על ההזדמנות שניתנה לי לעבוד על הפרויקט ומצפה להמשך הדרך.

# ביבליוגרפיה

1. “Circle-Circle Intersection” (2015).  *Wolfram Mathworld.* Wolfram Research. Retrieved August 21 2015, web address: http://mathworld.wolfram.com/Circle-CircleIntersection.html

2. Microsoft MSDN library (2015). Microsoft Corporation. Retrieved March 8 2015, web address: <https://msdn.microsoft.com/en-us/library/ms123401.aspx>.

3. Michael Dubson & Kathy Perkins. Geometric Optics (2015). PhET. Retrieved August 15 2015, web address: https://phet.colorado.edu/sims/geometric-optics/geometric-optics\_en.html

4. William Chu, 2001, “The Law of Refraction” ,Website: ‘Mathematics Department’. Retrieved November 21 2015, web address: <https://www.math.ubc.ca/~cass/courses/m309-01a/chu/Fundamentals/snell.htm>

5. William Chu , 2001, “Lenses” ,Website: ‘Mathematics Department’. Retrieved November 21 2015, web address: <https://www.math.ubc.ca/~cass/courses/m309-01a/chu/MirrorsLenses/lenses.htm>

6. ”Graphics Class”. Microsoft MSDN library (2015). Microsoft Corporation. Retrieved March 10 2015, web address: <https://msdn.microsoft.com/en-us/library/system.drawing.graphics(v=vs.110).aspx>

7. ” Rotate a point around another point” (2013) . Stackoverflow. Retrieved March 10 2015, web address: <http://stackoverflow.com/questions/13695317/rotate-a-point-around-another-point>

# תודות

ראשית, אני רוצה להודות לד"ר ולדימיר נודלמן על ההנחיה המסורה, ההסברים הברורים והמפורטים, על מתן מענה על כול שאלותיי והזמן והסבלנות הרבה לה הקדיש לעזור לי.

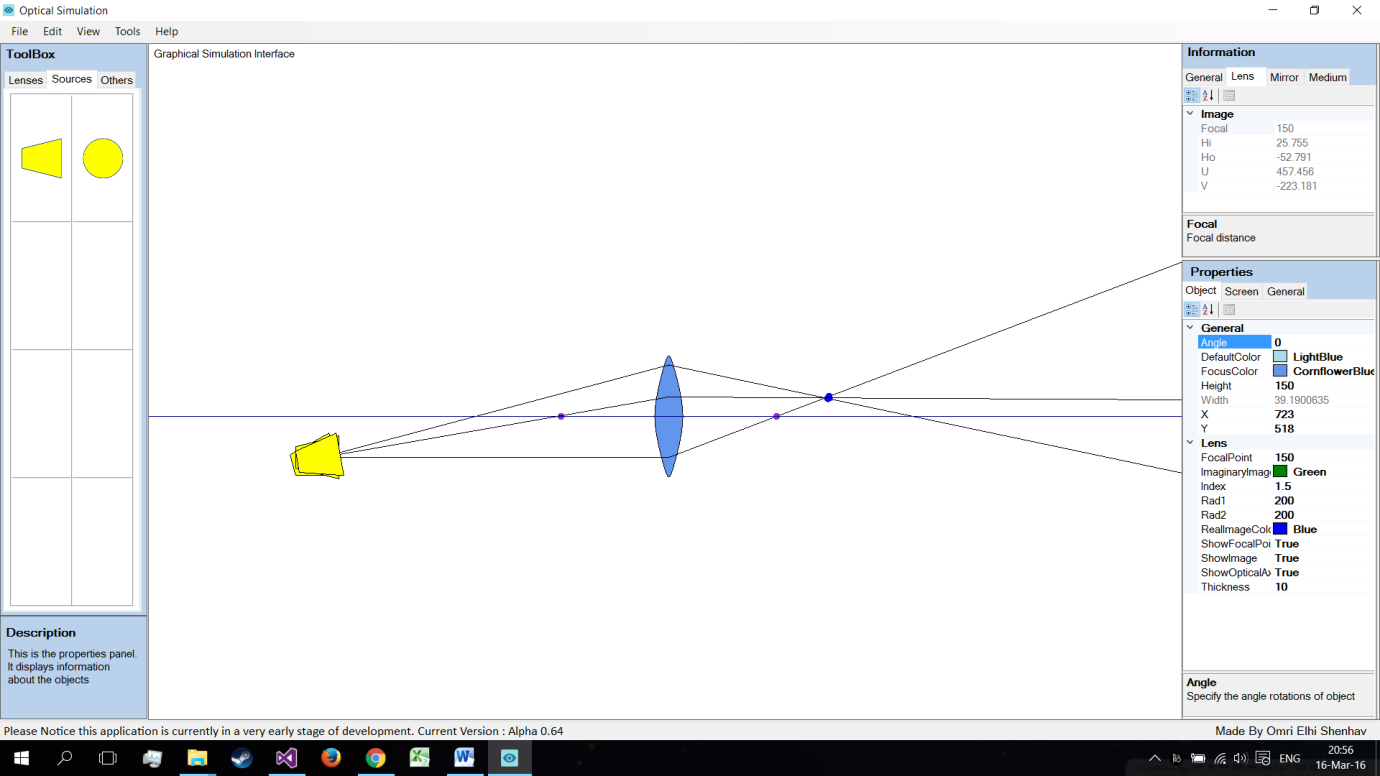
למורה המלווה שלי, ערן קצב, על שבזכותו הגעתי לתוכנית, ותמך בי וחברי מבית הספר שבתוכנית לאורך כול הדרך.

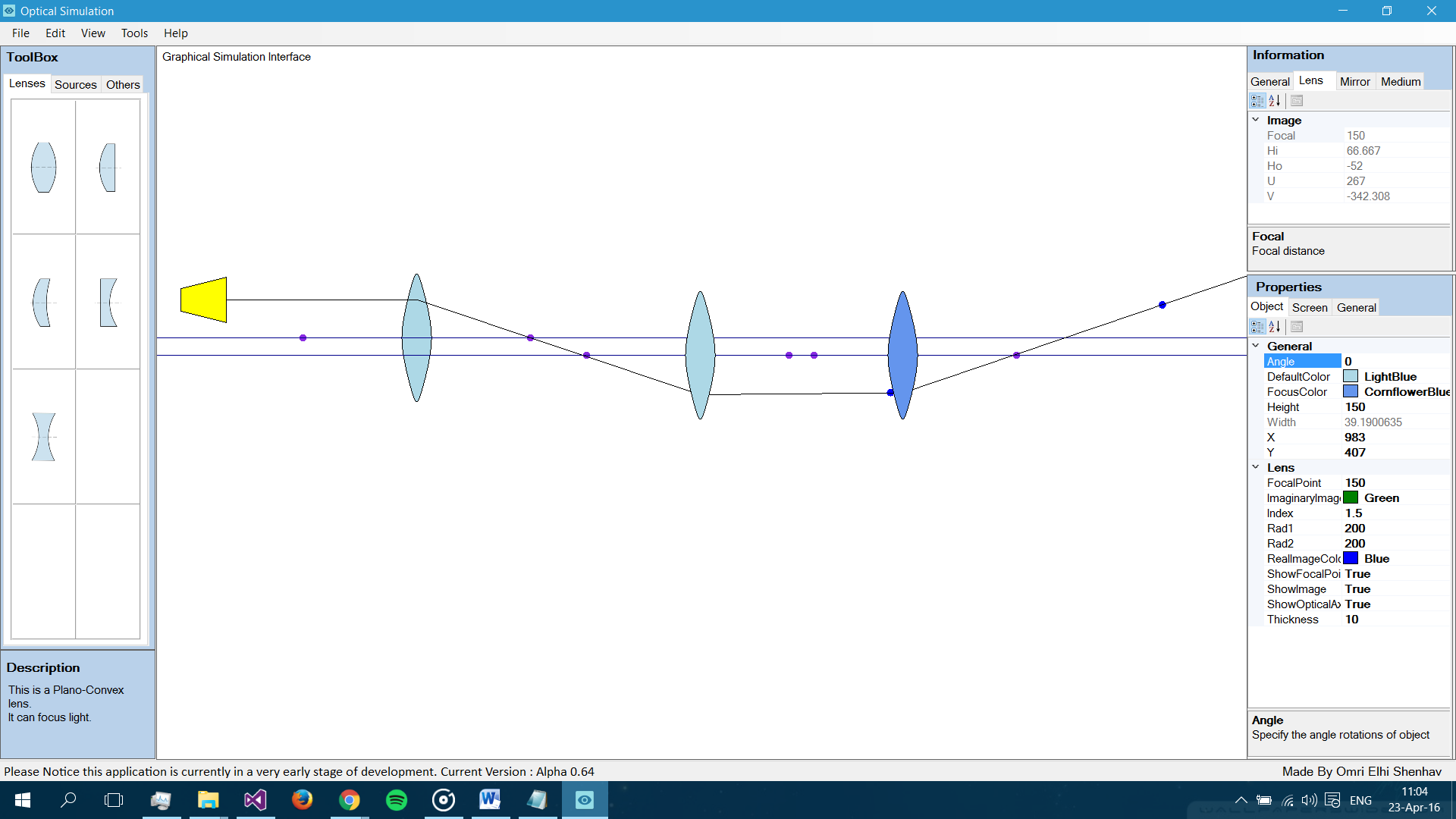
תודה כמובן למכון דוידסון שפתח בפני את האפשרות לקיים פרויקט ייחודי, מעניין ומשמעותי זה. אודה גם לצוות התוכנית 'מדעי המחשב אקדמיה ותעשיה' שתמכו בי לאורך כול התוכנית.

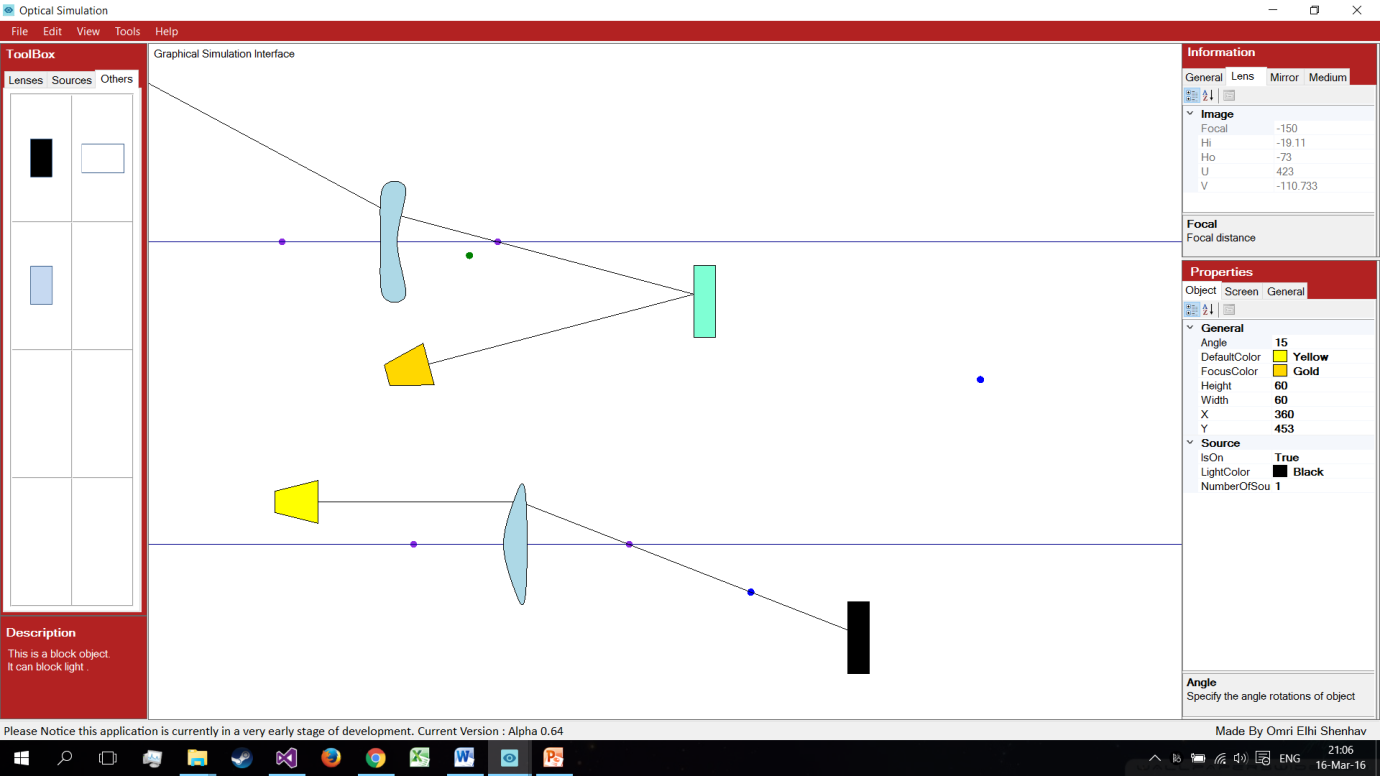
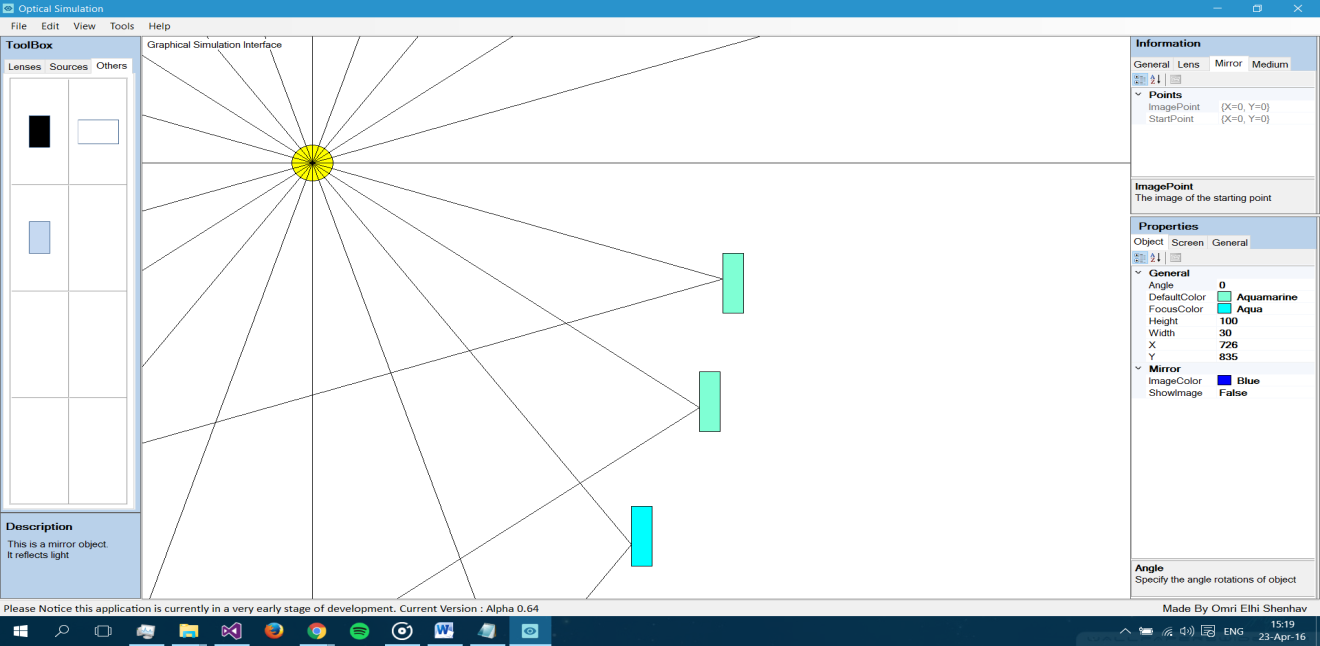
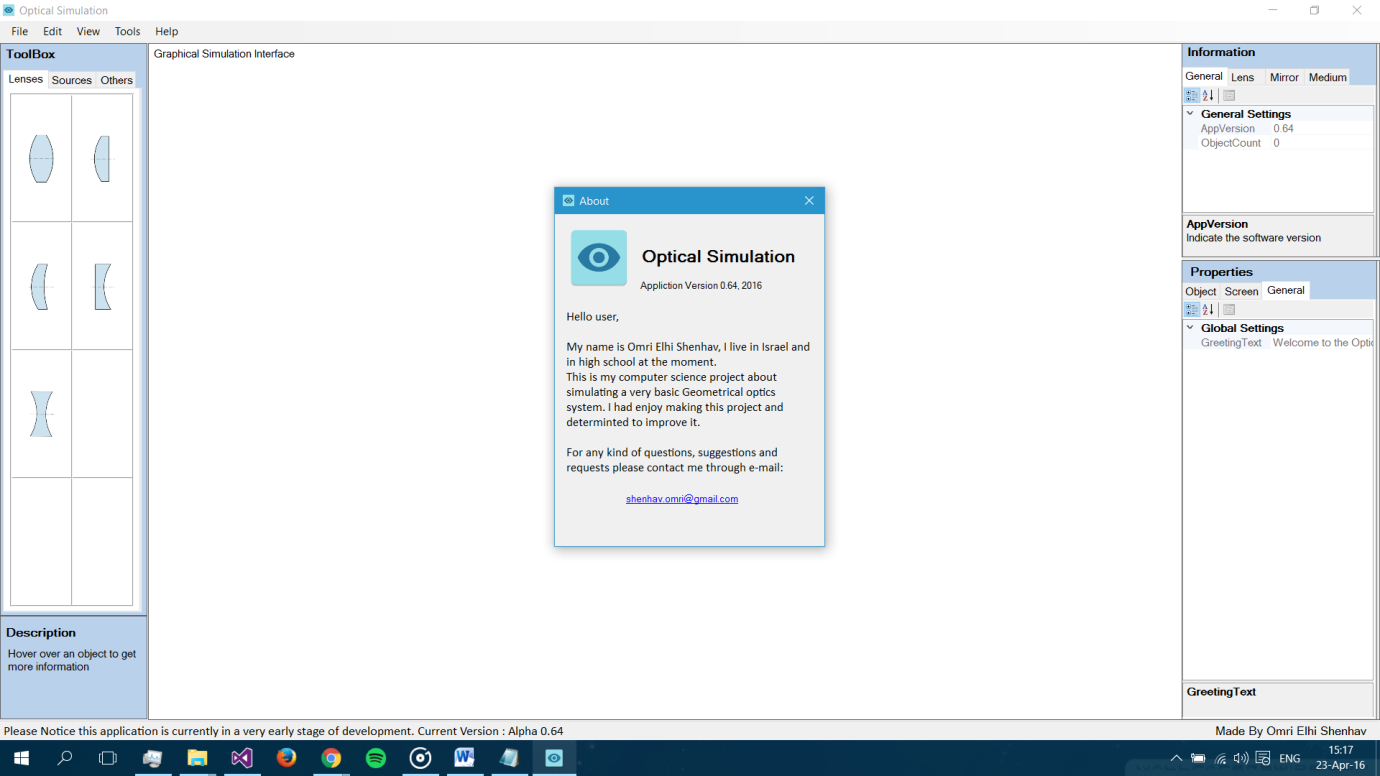
הוסיף להודות להורי על התמיכה והעזרה בהגהה הלשונית והעריכה.

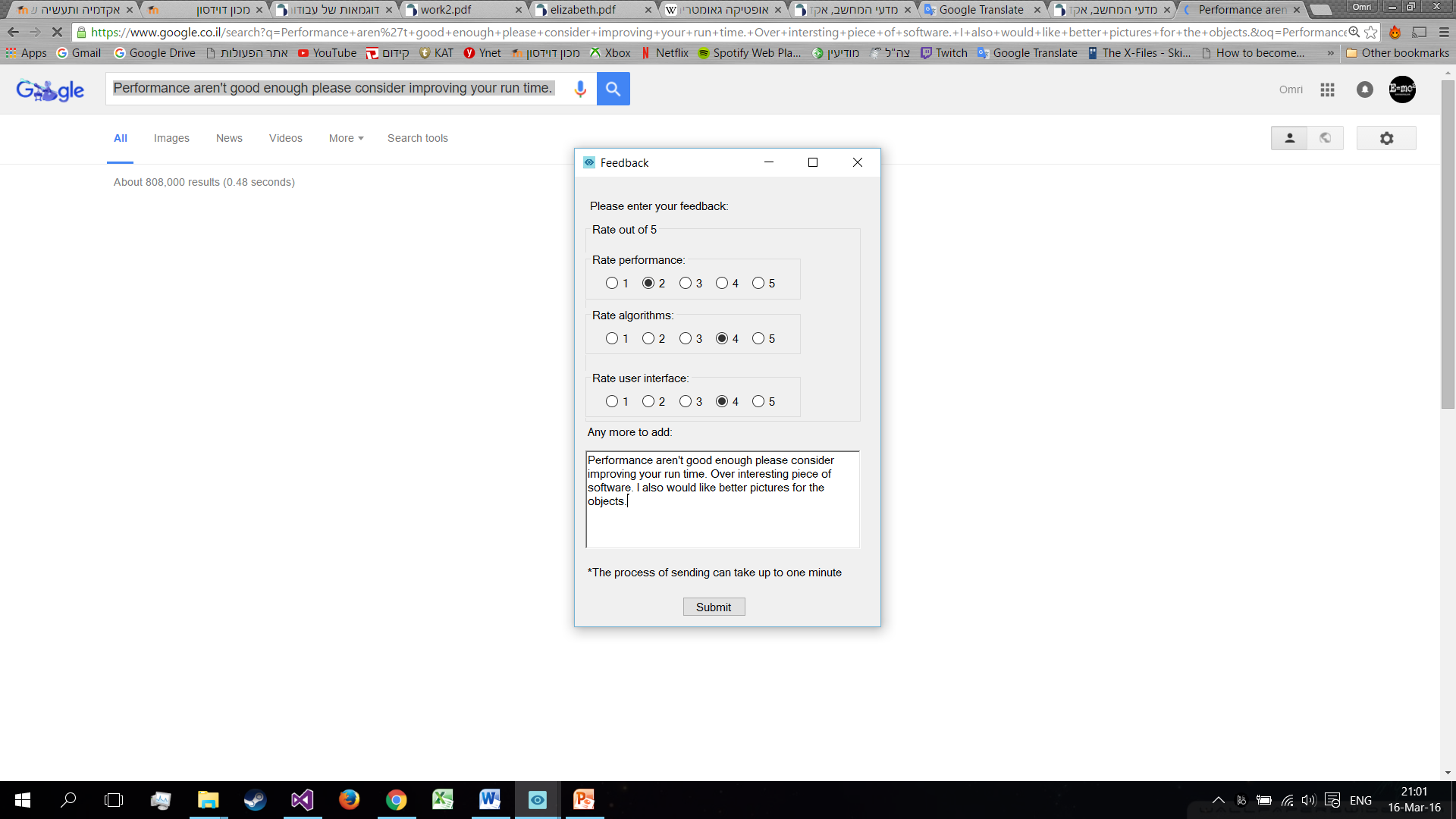
# נספחים

## דוגמאות מייצגות של הרצת המערכת









## תדפיס של הקוד המתועד

מחלקה: MathHelper

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Drawing.Drawing2D;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Runtime.InteropServices;

namespace Optical\_Simulation

{

//A class with static functions to help with Math and calculations

class MathHelper

{

//Input: Physical Object ref and PointF to rotate around the object center point

//Output: PointF with the new coordinates after the point has rotated around the object center point

public static PointF RotatePointF(PhysicalObject ob, PointF pointToRotate)

{

double angleInRadians = -ob.Angle \* (Math.PI / 180);

PointF centerPoint = new PointF(ob.X, ob.Y);

double cosTheta = Math.Cos(angleInRadians);

double sinTheta = Math.Sin(angleInRadians);

return new PointF

{

X =

(float)

(cosTheta \* (pointToRotate.X - centerPoint.X) -

sinTheta \* (pointToRotate.Y - centerPoint.Y) + centerPoint.X),

Y =

(float)

(sinTheta \* (pointToRotate.X - centerPoint.X) +

cosTheta \* (pointToRotate.Y - centerPoint.Y) + centerPoint.Y)

};

}

//Input: Physical Object ref and array of PointF to rotate around the object center point

//Output: PointF array each with the new coordinates after the point has rotated around the object center point

public static PointF[] RotatePointFArray(PhysicalObject ob, PointF[] p)

{

for (int i = 0; i < p.Length; i++)

{

p[i] = RotatePointF(ob, p[i]);

}

return p;

}

//Input: PointF to rotate around center point, center point to rotate around and angle or rotation

//Output: PointF with the new coordinates after the point has rotated around the object center point

public static PointF RotatePointF(PointF pointToRotate, PointF centerPoint, double angle)

{

double angleInRadians = -angle \* (Math.PI / 180);

double cosTheta = Math.Cos(angleInRadians);

double sinTheta = Math.Sin(angleInRadians);

return new PointF

{

X =

(float)

(cosTheta \* (pointToRotate.X - centerPoint.X) -

sinTheta \* (pointToRotate.Y - centerPoint.Y) + centerPoint.X),

Y =

(float)

(sinTheta \* (pointToRotate.X - centerPoint.X) +

cosTheta \* (pointToRotate.Y - centerPoint.Y) + centerPoint.Y)

};

}

//Input: Two PointF

//Output: The distance in double between the two points

public static double DistanceBetweenPointF(PointF p1, PointF p2)

{

return Math.Sqrt(Math.Pow(p1.X - p2.X, 2) + Math.Pow(p1.Y - p2.Y, 2));

}

//Input: Light and Graphics in which the light and the lens found at and index to ignore

//Output: return the closest object which is IReflect in the screen

public static int ClosestReflectionObject(Light light, Graphics g, int ignore)

{

int closesObj = -1;

double minDistance = Main\_Form.displayWidth, temp;

for (int i = 0; i < Main\_Form.objectsList.Count; i++)

{

if (Main\_Form.objectsList[i] is IInteract && i != ignore)

{

PhysicalObject obj = Main\_Form.objectsList[i];

if (((IInteract)obj).IsIntersect(light, g))

{

temp = MathHelper.DistanceBetweenPointF(light.pi, obj.CenterPoint);

if (temp < minDistance)

{

minDistance = temp;

closesObj = i;

}

}

}

}

return closesObj;

}

//Input: Light and Graphics in which the light and the lens found at

//Output: return the closest lens which the light hit or -1 if none were hit

public static int ClosestLens(Light light, Graphics g)

{

return ClosestReflectionObject(light, g, -1);

}

//Input: PointF in which the light start from and the angle or rotation of the light

//Output: PointF with the coordinate of the edge of the screen which this light hit

public static PointF GetEndLight(PointF start, double angle)

{

PointF end = new PointF(start.X, start.Y);

if (angle < 0) angle += 360;

angle = angle % 360;

if (angle == 0 || angle == 360) end.X = Main\_Form.displayWidth;

else if (angle == 90) end.Y = 0;

else if (angle == 180) end.X = 0;

else if (angle == 270) end.Y = Main\_Form.displayHeight;

else

{

double radius = Main\_Form.displayWidth + 100;

double y = radius \* Math.Sin(-angle \* Math.PI / 180) + start.Y;

double x = radius \* Math.Cos(-angle \* Math.PI / 180) + start.X;

end = new PointF((float)x, (float)y);

}

return end;

}

//Input: PointF to check if it's on an object in the objectList

//Output: If the PointF is on an object in the objectList return it's index if isn't one return -1

public static int DetectCollision(PointF point)

{

GraphicsPath path = new GraphicsPath();

try

{

if (Main\_Form.highlightedIndex != -1)

{

path = Main\_Form.objectsList[Main\_Form.highlightedIndex].GetPath();

if (path.IsVisible(point)) return Main\_Form.highlightedIndex;

}

for (int i = 0; i < Main\_Form.objectsList.Count; i++)

{

path = Main\_Form.objectsList[i].GetPath();

if (path.IsVisible(point)) return i;

}

return -1;

}

catch

{

return -1;

}

}

//Input: Two lights

//Output: Return the acute angle between the two lights

public static double AngleBetweenLights(Light a, Light b)

{

PointF v = new PointF(a.pf.X - a.pi.X, a.pf.Y - a.pi.Y);

PointF u = new PointF(b.pf.X - b.pi.X, b.pf.Y - b.pi.Y);

double num = Math.Abs(v.X \* u.X + v.Y \* u.Y);

num = num / ((Math.Sqrt(v.X \* v.X + v.Y \* v.Y) \* (Math.Sqrt(u.X \* u.X + u.Y \* u.Y))));

double angle = Math.Acos(num) \* 180 / Math.PI;

double correction = -0.3;

return Math.Round(angle + correction);

}

//Input: Two points from two Parallel Lines and the lines angle

//Output: Return distance between the lines

public static double DistanceBetweenParallelLines(PointF a, PointF b, double angle)

{

try

{

double slope = Math.Atan(angle);

double n1 = a.Y - slope \* a.X;

double n2 = b.Y - slope \* b.X;

double distance = Math.Abs(n1 - n2);

distance = distance / Math.Sqrt(1 + slope \* slope);

return distance;

}

catch { return 0; }

}

}

}

מחלקה: DrawingHelper

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Drawing.Drawing2D;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Runtime.InteropServices;

namespace Optical\_Simulation

{

//A class with static functions to help drawing

class DrawingHelper

{

//Input: Graphics g and list of Light

//Output: Draw stright line which alter with impact with IReflect objects

public static void DrawLightPath(Graphics g, List<Light> ray)

{

DrawLightPathHelper(g, ray, 0, -1);

}

//Input: Graphics g,list of light, number of times the function was repeated and index of object to ignore

//Output: Draw stright line which alter with impact with IReflect objects

public static void DrawLightPathHelper(Graphics g, List<Light> ray, int repeat, int ignore)

{

try

{

Light last = ray[ray.Count - 1];

int objIndex = MathHelper.ClosestReflectionObject(last, g, ignore);

if (objIndex != -1)

{

if (Main\_Form.objectsList[objIndex] is Lens)

{

if (((Lens)Main\_Form.objectsList[objIndex]).ShowFocalPoints || ((Lens)Main\_Form.objectsList[objIndex]).ShowOpticalAxis)

DrawFocalPointsAndOpticalAxis(g, (Lens)Main\_Form.objectsList[objIndex]);

}

((IInteract)Main\_Form.objectsList[objIndex]).Interact(ray, g); //change the course of the light

//this part check if the light will hit another lens, if so it will recall this function (recursively)

bool check=true;

if (Main\_Form.objectsList[objIndex] is Block) check=false;

int temp = objIndex; //which object to ignore

last = ray[ray.Count - 1];

objIndex = MathHelper.ClosestReflectionObject(last, g,temp);

if (objIndex != -1 && repeat<Main\_Form.maxInteractions-1 &&check)

{

repeat++;

DrawLightPathHelper(g, ray, repeat, temp);

}

else

{

for (int h = 0; h < ray.Count; h++)

{

g.DrawLine(new Pen(ray[h].LightColor), ray[h].pi, ray[h].pf);

}

}

}

else

{

for (int h = 0; h < ray.Count; h++)

{

g.DrawLine(new Pen(ray[h].LightColor), ray[h].pi, ray[h].pf);

}

}

}

catch

{

//MessageBox.Show(ex.Message);

}

}

//Input: Graphics g and a lens

//Output: Draw the either the focal points of the lens or/and the lens optical axis

public static void DrawFocalPointsAndOpticalAxis(Graphics g, Lens lens)

{

//draw the focal points of the lens who got hit

PointF focal1 = new PointF(lens.X + (float)lens.FocalPoint - 5, lens.Y - 5);

PointF focal2 = new PointF(lens.X - (float)lens.FocalPoint - 5, lens.Y - 5);

focal1 = MathHelper.RotatePointF(lens,focal1 );

focal2 = MathHelper.RotatePointF(lens, focal2);

if (lens.ShowFocalPoints)

{

g.FillEllipse(Brushes.BlueViolet, focal1.X, focal1.Y, 10, 10);

g.FillEllipse(Brushes.BlueViolet, focal2.X, focal2.Y, 10, 10);

}

if(lens.ShowOpticalAxis)

{

Light l1 = new Light(lens.CenterPoint, MathHelper.GetEndLight(lens.CenterPoint, lens.Angle));

Light l2 = new Light(lens.CenterPoint, MathHelper.GetEndLight(lens.CenterPoint, lens.Angle+180));

g.DrawLine(Pens.DarkBlue, l1.pi, l1.pf);

g.DrawLine(Pens.DarkBlue, l2.pi, l2.pf);

}

}

}

}

מחלקה: AlgoritemHelper

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Drawing.Drawing2D;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Runtime.InteropServices;

namespace Optical\_Simulation

{

//A class with static functions to help with the main optical algoritems

class AlgoritemHelper

{

//Input: Light list, graphics g and a Lens

//Output: Change the Light list to the path casued by the lens

public static void LensAlgoritem(List<Light> lightList, Graphics g, Lens obj)

{

try

{

Light last = lightList[lightList.Count - 1];

PointF pi = last.pi;

PointF pf = last.pf;

PointF firstPoint = new PointF(pi.X, pi.Y);

Color lightColor = last.LightColor;

double initialAngle = last.GetAngle();

Region lensReg = obj.GetRegion();

Region rayReg = last.GetRegion();

lensReg.Intersect(rayReg); //Intersect area of lens and the last light

if (!lensReg.IsEmpty(g))

{

//getting the hit point

RectangleF boundsRect = lensReg.GetBounds(g);

PointF hit = new PointF((boundsRect.Right + boundsRect.Left) / 2, (boundsRect.Top + boundsRect.Bottom) / 2);

pf = hit;

//creating a light bewtween initial point and hit point

lightList[lightList.Count - 1] = new Light(pi, pf, lightColor);

pi = pf;

Light light = new Light(pi,pf);

double beta = 0;

beta = GetImagePointAngle(g, firstPoint, obj, initialAngle, hit);

//creating a light between the hit point and on

light = new Light(hit, MathHelper.GetEndLight(hit, beta),lightColor);

lightList.Add(light);

}

}

catch

{

}

}

//Input: Graphics g, point which the last light start from, Lens object, angle of last light and the hit point

//Output: calculate the angle in which the light comes out of the lens

public static double GetImagePointAngle(Graphics g, PointF firstPoint, Lens obj, double angle, PointF hit)

{

bool imaginary = false;

float f = (float)obj.FocalPoint;

float u = (float)Math.Abs(obj.X - firstPoint.X);

float ho = firstPoint.Y - obj.Y;

float v = 1 / ((1 / f) - (1 / u));

float hi = (v \* ho / u);

PointF imagePoint = new PointF();

bool isVertical = obj.Angle > 225 && obj.Angle < 315 || obj.Angle > 45 && obj.Angle < 135;

if (!isVertical)

{

Color imageColor = obj.RealImageColor;

if (v \* u < 0)

{

imageColor = obj.ImaginaryImageColor;

imaginary = true;

}

if (firstPoint.X > obj.X) v \*= -1;

imagePoint = new PointF(obj.X + (float)(v), obj.Y - (float)hi);

/\*double angleToRotate = obj.Angle;

if (angleToRotate >= 90 && angleToRotate <= 270) angleToRotate = angleToRotate + 180;

imagePoint = MathHelper.RotatePointF(imagePoint, obj.CenterPoint, 0\*angleToRotate);\*/

//drawing image

if (obj.ShowImage) g.FillEllipse(new SolidBrush(imageColor), imagePoint.X - 5, imagePoint.Y - 5, 10, 10);

}

else

{

u = (float)Math.Abs(obj.Y - firstPoint.Y);

ho = -firstPoint.X + obj.X;

v = 1 / ((1 / f) - (1 / u));

hi = (v \* ho / u);

Color imageColor = obj.RealImageColor;

if (v \* u < 0)

{

imageColor = obj.ImaginaryImageColor;

imaginary = true;

}

if (firstPoint.Y > obj.Y) v \*= -1;

imagePoint = new PointF(obj.X + (float)(v), obj.Y - (float)hi);

/\*double angleToRotate = obj.Angle;

if (angleToRotate >= 0 && angleToRotate <= 180) angleToRotate = angleToRotate + 180;

imagePoint = MathHelper.RotatePointF(imagePoint, obj.CenterPoint, angleToRotate);\*/

//drawing image

if (obj.ShowImage) g.FillEllipse(new SolidBrush(imageColor), imagePoint.X - 5, imagePoint.Y - 5, 10, 10);

}

//give information to data class

if (Main\_Form.highlightedIndex != -1)

{

PhysicalObject highlighted = Main\_Form.objectsList[Main\_Form.highlightedIndex];

if (highlighted == obj)

{

LensInfo.focal = Math.Round(f, 3);

LensInfo.u = Math.Round(u, 3);

LensInfo.v = Math.Round(-v, 3);

LensInfo.ho = Math.Round(-ho, 3);

LensInfo.hi = Math.Round(hi, 3);

}

}

double angleToReturn = new Light(hit, imagePoint).GetAngle();

if (imaginary) angleToReturn = new Light(hit, imagePoint).GetAngle() + 180;

return angleToReturn;

}

//Input: Light list, graphics g and a Physical object

//Output: Change the Light list to the path casued by the Mirror

public static void MirrorAlgoritem(List<Light> lightList, Graphics g, PhysicalObject obj)

{

Light last = lightList[lightList.Count - 1];

PointF pi = last.pi;

PointF pf = last.pf;

PointF firstPoint = new PointF(pi.X, pi.Y);

Color lightColor = last.LightColor;

Region mirrorReg = obj.GetRegion();

Region rayReg = last.GetRegion();

double angle = last.GetAngle();

bool vertical = false;

mirrorReg.Intersect(rayReg); //Intersect area of lens and the last light

if (!mirrorReg.IsEmpty(g))

{

PointF hit = GetHitPoint(pf, mirrorReg, g);

lightList[lightList.Count - 1] = new Light(pi, hit, lightColor); //light from the start to the hit point

double alpha = 180 - (angle - 2 \* obj.Angle);

//check if the light hiting the vertical part of the mirror

PointF[] points = obj.GetPoints();

if (points.Length == 4)

{

PointF p1 = points[0];

PointF p2 = points[1];

PointF p3 = points[2];

PointF p4 = points[3];

float x = hit.X;

float y = hit.Y;

bool check1 = x >= Math.Min(p1.X, p4.X) && x <= Math.Max(p1.X, p4.X) && y >= Math.Min(p1.Y, p4.Y) && y <= Math.Max(p1.Y, p4.Y);

bool check2 = x >= Math.Min(p2.X, p3.X) && x <= Math.Max(p2.X, p3.X) && y >= Math.Min(p2.Y, p3.Y) && y <= Math.Max(p2.Y, p3.Y);

vertical = check1 || check2;

if (vertical)

{

alpha += 180;

}

}

lightList.Add(new Light(hit, MathHelper.GetEndLight(hit, alpha), lightColor)); //light from the hit point farther

//draw image and give info

Color imageColor = Color.Blue;

Mirror mirror = obj as Mirror;

if (mirror != null&&mirror.ShowImage)

{

//calc image

imageColor = mirror.ImageColor;

double beta = alpha + 180;

double radius = MathHelper.DistanceBetweenPointF(hit, firstPoint);

float height = (float)(radius \* Math.Sin(beta \* Math.PI / 180));

float width = (float)(radius \* Math.Cos(beta \* Math.PI / 180));

float addWidth = obj.X - hit.X;

float addHeight = obj.Y - hit.Y;

//

//draw image

PointF imagePoint = new PointF(obj.X + width + addWidth, obj.Y - height + addHeight);

double angleToRotate = obj.Angle;

if (angleToRotate >= 180 && angleToRotate <= 360) angleToRotate = angleToRotate + 180;

imagePoint = MathHelper.RotatePointF(imagePoint, obj.CenterPoint, angleToRotate);

g.FillEllipse(new SolidBrush(imageColor), imagePoint.X - 5, imagePoint.Y - 5, 10, 10);

//

//give information to data class

if (Main\_Form.highlightedIndex != -1)

{

PhysicalObject highlighted = Main\_Form.objectsList[Main\_Form.highlightedIndex];

if (highlighted == obj)

{

MirrorInfo.startPoint = firstPoint;

//MirrorInfo.imagePoint = imagePoint;

}

}

}

//

}

}

//Input: Light list, graphics g, Physical object, screen index, medium index

//Output: Change the Light list to the path casued by the Medium

public static void MediumAlgoritem(List<Light> lightList, Graphics g, PhysicalObject obj,double n1,double n2)

{

Light last = lightList[lightList.Count - 1];

PointF pi = last.pi;

PointF pf = last.pf;

PointF firstPoint = new PointF(pi.X, pi.Y);

Color lightColor = last.LightColor;

Region mediumReg = obj.GetRegion();

Region rayReg = last.GetRegion();

double angle = last.GetAngle();

mediumReg.Intersect(rayReg); //Intersect area of lens and the last light

if (!mediumReg.IsEmpty(g))

{

PointF hit = GetHitPoint(pf, mediumReg, g);

bool oppSide = false, isCritical=false;

double alpha = 0,beta=0,tempBeta=0,criticalAngle=0;//angles

//getting hiting angle using math

PointF[] points = obj.GetPoints();

Light first = new Light(firstPoint, hit);

Light second = new Light(firstPoint, hit);//need to modified

if (points.Length == 2) second = new Light(points[0], points[1]);

alpha = 90 - MathHelper.AngleBetweenLights(first, second);

//

if ((firstPoint.X - hit.X) \* (obj.X - hit.X) > 0) oppSide = true;

//calculate critical angle

if (!oppSide)

{

if (n1 > n2) criticalAngle = Math.Round(Math.Asin(n2 / n1) \* 180 / Math.PI);

isCritical = criticalAngle != 0 && alpha >= criticalAngle;

}

else

{

if (n2 > n1) criticalAngle = Math.Round(Math.Asin(n1 / n2) \* 180 / Math.PI);

isCritical = criticalAngle != 0 && alpha >= criticalAngle;

}

//

if (isCritical)//act like a mirror

{

MirrorAlgoritem(lightList, g, obj);

}

else

{

if (!oppSide)

{

beta = (Math.Asin(n1 / n2 \* Math.Sin(alpha \* Math.PI / 180)) \* 180 / Math.PI);//snal law

beta = Math.Round(beta);

tempBeta = beta;//save beta value

}

else //opposite side

{

beta = (Math.Asin(n2 / n1 \* Math.Sin(alpha \* Math.PI / 180)) \* 180 / Math.PI);//snal law

tempBeta = Math.Round(beta);//save beta value

beta = 180 - Math.Round(beta);

}

//fix light direction

float dirction = (firstPoint.Y - hit.Y) / (firstPoint.X - hit.X);

dirction = Math.Abs(dirction) / dirction;

if (oppSide) dirction \*= -1;

if (dirction == 1) beta = 360 - beta;

//

if (alpha % 90 != 0)

{

lightList[lightList.Count - 1] = new Light(firstPoint, hit, lightColor);//light from the start to the hit point

lightList.Add(new Light(hit, MathHelper.GetEndLight(hit, beta + obj.Angle), lightColor));

}

}

//give information to data class

if (Main\_Form.highlightedIndex != -1)

{

PhysicalObject highlighted = Main\_Form.objectsList[Main\_Form.highlightedIndex];

if (highlighted == obj)

{

MediumInfo.alphaAngle = alpha;

MediumInfo.betaAngle = tempBeta;

MediumInfo.mediumIndex = n2;

MediumInfo.criticalAngle = criticalAngle;

}

}

}

}

//Input: the end PointF of a light, reg of the Physical object, Graphics of screen

//Output: Get a hit point in the center of the object (if hit)

public static PointF GetHitPoint(PointF pf, Region reg, Graphics g)

{

//getting the heat point

RectangleF boundsRect = reg.GetBounds(g);

PointF hit = new PointF();

if (Math.Abs(boundsRect.Top - pf.Y) > Math.Abs(boundsRect.Bottom - pf.Y)) hit.Y = boundsRect.Top;

else hit.Y = boundsRect.Bottom;

if (Math.Abs(boundsRect.Right - pf.X) > Math.Abs(boundsRect.Left - pf.X)) hit.X = boundsRect.Right;

else hit.X = boundsRect.Left;

return hit;

}

//Input: Light list, graphics g, Physical object

//Output: Change the Light list to the path casued by the Block

public static void BlockAlgoritem(List<Light> lightList, Graphics g, PhysicalObject obj)

{

Light last = lightList[lightList.Count - 1];

PointF pi = last.pi;

PointF pf = last.pf;

Color lightColor = last.LightColor;

Region BlockReg = obj.GetRegion();

Region rayReg = last.GetRegion();

BlockReg.Intersect(rayReg); //Intersect area of lens and the last light

if (!BlockReg.IsEmpty(g))

{

//getting the heat point

RectangleF boundsRect = BlockReg.GetBounds(g);

PointF hit = new PointF((boundsRect.Right + boundsRect.Left) / 2, (boundsRect.Top + boundsRect.Bottom) / 2);

lightList[lightList.Count - 1] = new Light(pi, hit, lightColor); //light from start to hit point

}

}

}

}

מחלקה: IDrawable

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Drawing.Drawing2D;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Runtime.InteropServices;

namespace Optical\_Simulation

{

//interface for object which could be drawn

interface IDrawable

{

PointF[] GetPoints();

Region GetRegion();

GraphicsPath GetPath();

void Draw(Graphics g, bool highlighted);

}

}

מחלקה: IInteract

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Drawing.Drawing2D;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Runtime.InteropServices;

namespace Optical\_Simulation

{

//interface of objects which can alter light direction

interface IInteract

{

void Interact(List<Light> lightList, Graphics g);

bool IsIntersect(Light ray, Graphics g);

}

}

מחלקה: ISource

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Drawing.Drawing2D;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Runtime.InteropServices;

namespace Optical\_Simulation

{

//interface of objects which can be used as a Light source

interface ISource

{

List<Light> EmitLight(PointF start, double angle);

}

}

מחלקה: PhysicalObject

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Drawing.Drawing2D;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Runtime.InteropServices;

namespace Optical\_Simulation

{

[Serializable]

public abstract class PhysicalObject:IDrawable

{

protected float width,height;//width and height of the object

protected float x, y;//x and y coordinates of the object

protected double angle;//angle of the object

protected Color defaultColor, focusColor;//colors of the object

/// <summary>

/// Empty constructor for PhysicalObject, gives defult values.

/// </summary>

public PhysicalObject()

{

}

public virtual Object GetCopy() //should be override in non abstract classes

{

return null;

}

//abstract functions

public abstract PointF[] GetPoints();

public abstract Region GetRegion();

public abstract GraphicsPath GetPath();

public abstract void Draw(Graphics g, bool highlighted);

//

[Category("General"), Description("Default Color of the object")]

public virtual Color DefaultColor

{

set { this.defaultColor = value; }

get { return this.defaultColor; }

}

[Category("General"), Description("The color of the object when it's focused")]

public virtual Color FocusColor

{

set { this.focusColor = value; }

get { return this.focusColor; }

}

[Category("General"), Description("The X axis value")]

public virtual float X

{

set { if (value >= 0 && value <= Main\_Form.displayWidth)this.x = value; else UserValuesWarning(0, Main\_Form.displayWidth); }

get { return this.x; }

}

[Category("General"), Description("The Y axis value")]

public virtual float Y

{

set { if (value >= 0 && value <= Main\_Form.displayHeight)this.y = value; else UserValuesWarning(0, Main\_Form.displayHeight); }

get { return this.y; }

}

[Category("General"), Description("Specify the Width of object")]

public virtual float Width

{

set { if (value >= 0 && value <= Main\_Form.displayWidth)this.width = value; else UserValuesWarning(0, Main\_Form.displayWidth); }

get { return this.width; }

}

[Category("General"), Description("Specify the Height of object")]

public virtual float Height

{

set { if (value >= 0 && value <= Main\_Form.displayHeight)this.height = value; else UserValuesWarning(0, Main\_Form.displayHeight); }

get { return this.height; }

}

[Category("General"), Description("Specify the angle rotations of object")]

public virtual double Angle

{

get

{

return angle;

}

set

{

while (value < 0) value += 360;

angle = value%360;

}

}

[Browsable(false)]

public virtual PointF CenterPoint

{

get

{

return new PointF(this.X,this.Y);

}

set

{

this.X = value.X;

this.Y = value.Y;

}

}

public virtual void UserValuesWarning(double min,double max)

{

MessageBox.Show("Values should be between " + min + "-" + max);

}

}

}

מחלקה: Light

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Drawing.Drawing2D;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Runtime.InteropServices;

using System.Numerics;

using System.Windows;

namespace Optical\_Simulation

{

[Serializable]

public sealed class Light

{

public PointF pi, pf;//two points which define the line

private Color lightColor;//the light color

/// <summary>

/// Light constrctor with color

/// Input: initial PointF of light, finle PoinF of light and Light Color

/// </summary>

public Light(PointF pi, PointF pf, Color lightColor)

{

this.pi = pi;

this.pf = pf;

this.LightColor = lightColor;

}

/// <summary>

/// Light constrctor without color (gives defult color)

/// Input: initial PointF of light, finle PoinF of light

/// </summary>

public Light(PointF pi, PointF pf) : this(pi, pf, Color.Black) { }

public Color LightColor { set { this.lightColor = value; } get { return this.lightColor; } }

/// <summary>

/// return the area near the light for calculations

/// </summary>

public Region GetRegion()

{

GraphicsPath path = new GraphicsPath();

PointF p1, p2, p3, p4;

if (this.GetAngle() >= 315 && this.GetAngle() <= 0 || this.GetAngle() >= 0 && this.GetAngle() <= 45 || this.GetAngle() >= 135 && this.GetAngle() <= 225)

{

p1 = new PointF(pi.X, pi.Y + 1);

p2 = new PointF(pf.X, pf.Y + 1);

p3 = new PointF(pi.X, pi.Y - 1);

p4 = new PointF(pf.X, pf.Y - 1);

}

else

{

p1 = new PointF(pi.X + 1, pi.Y);

p2 = new PointF(pf.X + 1, pf.Y);

p3 = new PointF(pi.X - 1, pi.Y);

p4 = new PointF(pf.X - 1, pf.Y);

}

path.AddLine(p1, p2);

path.AddLine(p2, p4);

path.AddLine(p4, p3);

path.AddLine(p3, p1);

path.CloseAllFigures();

return new Region(path);

}

/// <summary>

/// return the angle of the light in double

/// </summary>

public double GetAngle()

{

double angle = 0;

double xDiff = pf.X - pi.X;

double yDiff = pi.Y - pf.Y;

if (Math.Abs(xDiff) < 0.000001) xDiff = xDiff/Math.Abs(xDiff)\*0.1;

if (Math.Abs(yDiff) < 0.000001) yDiff = yDiff / Math.Abs(yDiff) \* 0.1;

double slope = yDiff / xDiff;

if (pf.Y == pi.Y && pi.X < pf.X) angle = 0;

else if (pf.X == pi.X && pf.Y < pi.Y) angle = 90;

else if (pf.Y == pi.Y && pi.X > pf.X) angle = 180;

else if (pf.X == pi.X && pf.Y > pi.Y) angle = 270;

else angle = Math.Atan2(yDiff, xDiff) \* (180 / Math.PI);

if (angle < 0) angle += 360;

return angle % 360;

}

}

}

מחלקה: Lens

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Drawing.Drawing2D;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Runtime.InteropServices;

namespace Optical\_Simulation

{

[Serializable]

public class Lens : PhysicalObject, IInteract

{

protected double radius1, radius2;//radii of the lens

/// <summary>

/// Empty constructor for Lens, gives defult values.

/// </summary>

public Lens(int type)

{

this.Type = type; //The type of lens is accourding to the pictures seleced in the toolbox

SetDefaults(); //defualt values for all lenses

}

public override object GetCopy()

{

Lens obj = new Lens(this.Type);

obj.Angle = this.angle;

obj.defaultColor = this.defaultColor;

obj.focusColor = this.focusColor;

obj.height = this.height;

obj.width = this.width;

obj.x = this.x;

obj.y = this.y;

obj.radius1 = this.radius1;

obj.radius2 = this.radius2;

return obj;

}

public void SetDefaults()

{

this.Index = 1.5;

this.Height = 150;

this.Width = 20;

this.Thickness = 20;

this.DefaultColor = Color.LightBlue;

this.FocusColor = Color.CornflowerBlue;

this.ShowImage = true;

this.ShowOpticalAxis = true;

this.ShowFocalPoints = true;

this.RealImageColor = Color.Blue;

this.ImaginaryImageColor = Color.Green;

SetDefultsForEachType();

}

//defualt values for each type of lens

public void SetDefultsForEachType()

{

if(this.Type== 1)

{

this.FocalPoint = 150;

this.Rad1 = 200;

this.Rad2 = 200;

this.Thickness = 10;

}

if (this.Type == 2)

{

this.FocalPoint = 150;

this.Rad1 = 1000;

this.Rad2 = 150;

this.Thickness = 10;

}

if (this.Type == 3)

{

this.FocalPoint = 150;

this.Rad1 = -300;

this.Rad2 = 150;

this.Thickness = 10;

}

if (this.Type == 4)

{

this.FocalPoint = -150;

this.Rad1 = -250;

this.Rad2 = 1000;

this.Thickness = 20;

}

if (this.Type == 5)

{

this.FocalPoint = -150;

this.Rad1 = -250;

this.Rad2 = -250;

this.Thickness = 20;

}

}

//return array of points describing the lens

public override PointF[] GetPoints()

{

float halfHeight =this.Height/2,halfWidth=this.Width/2;

double r1 = this.Rad1, r2 = this.Rad2;

PointF[] pointsArr = new PointF[6];

PointF[] radius1Points = PointsByRadius(r1, this.height / 2,true);

PointF[] radius2Points = PointsByRadius(r2, this.height / 2,false);

pointsArr[0] = radius1Points[0];

pointsArr[1] = radius1Points[1];

pointsArr[2] = radius1Points[2];

pointsArr[3] = radius2Points[0];

pointsArr[4] = radius2Points[1];

pointsArr[5] = radius2Points[2];

float width = (float)MathHelper.DistanceBetweenPointF(pointsArr[1], pointsArr[4]);

float lowWidth =(float)MathHelper.DistanceBetweenPointF(pointsArr[0], pointsArr[5]);

if (width > lowWidth)

this.FocalPoint = Math.Abs(this.FocalPoint);

else this.FocalPoint = -1\*Math.Abs(this.FocalPoint);

this.Width = width;

pointsArr = MathHelper.RotatePointFArray(this,pointsArr);

return pointsArr;

}

//using math the function returnes array of 3 points representing half ellipse which is half the lens

private PointF[] PointsByRadius(double radius,float height,bool isR1)

{

PointF[] arr = new PointF[3]; //array of 3 points which describe half ellipse according to the radius

float distance = (float)(Math.Abs(radius) - Math.Sqrt(radius \* radius - height \* height)); //width of the half ellipse using math

float thick = (float)(this.Thickness / 2); //thickness added to each side of the half ellipse

if (isR1) //radius 1

{

if (radius < 0)

{

arr[0] = new PointF(this.X + distance + thick, this.Y - height);

arr[1] = new PointF(this.X + thick, this.Y);

arr[2] = new PointF(this.X + distance + thick, this.Y + height);

}

else

{

arr[0] = new PointF(this.X + thick, this.Y - height);

arr[1] = new PointF(this.X + thick + distance, this.Y);

arr[2] = new PointF(this.X + thick, this.Y + height);

}

}

else //radius 2

{

if (radius < 0)

{

arr[2] = new PointF(this.X - distance - thick, this.Y - height);

arr[1] = new PointF(this.X - thick, this.Y);

arr[0] = new PointF(this.X - distance - thick, this.Y + height);

}

else

{

arr[2] = new PointF(this.X - thick, this.Y - height);

arr[1] = new PointF(this.X - distance - thick, this.Y);

arr[0] = new PointF(this.X - thick, this.Y + height);

}

}

return arr;

}

//returnes the lens Region

public override Region GetRegion()

{

return new Region(this.GetPath());

}

//return the lens GraphicsPath

public override GraphicsPath GetPath()

{

GraphicsPath path = new GraphicsPath();

path.AddClosedCurve(this.GetPoints());

return path;

}

//Draw the lens using the colors in the variables and also a black line around the object

public override void Draw(Graphics g, bool highlighted)

{

Region reg = this.GetRegion();

SolidBrush brushDefault = new SolidBrush(this.DefaultColor);

SolidBrush brushFocus = new SolidBrush(this.FocusColor);

if (highlighted) g.FillRegion(brushFocus, reg);

else g.FillRegion(brushDefault, reg);

g.DrawPath(Pens.Black, this.GetPath());

}

//override the Height property in order to change the radii together with the height

[Category("General"), Description("Specify the Height of object")]

public override float Height

{

set

{

radius1 += radius1 / Math.Abs(radius1)\*(value - height) / 2;

radius2 += radius2 / Math.Abs(radius2) \* (value - height) / 2;

height = value;

}

get

{

return height;

}

}

//override the Width property in order to keep the user from changing it

[Category("General"), Description("Specify the Width of object"),ReadOnly(true)]

public override float Width

{

set { if (value >= 0 && value <= Main\_Form.displayWidth)this.width = value; else UserValuesWarning(0, Main\_Form.displayWidth); }

get { return this.width; }

}

[Browsable(false)]

//the type of lens

public int Type { set; get; }

[Category("Lens"), Description("Determine whether or not to show the image created by the lens or not")]

public bool ShowImage { set; get; }

[Category("Lens"), Description("Determine whether or not to show the focal points of the lens when it is hitted")]

public bool ShowFocalPoints { set; get; }

[Category("Lens"), Description("Determine whether or not to show the optical axis of the lens")]

public bool ShowOpticalAxis { set; get; }

[Category("Lens"), Description("Determine the color of the real image created by the lens")]

public Color RealImageColor { set; get; }

[Category("Lens"), Description("Determine the color of the imaginary image created by the lens")]

public Color ImaginaryImageColor { set; get; }

[Category("Lens"), Description("Focal point of the lens")]

public double FocalPoint { set; get; }

[Category("Lens"), Description("Radius number 1 of the lens")]

public double Rad1

{

get

{

return radius1;

}

set

{

if (Math.Abs(value) > this.Height / 2) //keep the user from giving impossible values

radius1 = value;

else MessageBox.Show("impossible lens");

}

}

[Category("Lens"), Description("Radius number 2 of the lens")]

public double Rad2

{

get

{

return radius2;

}

set

{

if (Math.Abs(value) > this.Height / 2)//keep the user from giving impossible values

radius2 = value;

else MessageBox.Show("impossible lens");

}

}

[Category("Lens"), Description("Index of the lens")]

public double Index { set; get; }

[Category("Lens"), Description("Thickness added to the lens")]

public double Thickness { set; get; }

[Browsable(false)]

//The function check if a light ray hit the lens using the region of both

public bool IsIntersect(Light ray, Graphics g)

{

Region lensReg = this.GetRegion();

Region rayReg = ray.GetRegion();

lensReg.Intersect(rayReg);

if (!lensReg.IsEmpty(g)) return true;

return false;

}

//The function will make the light bend accordingly if the light hit it

public void Interact(List<Light> lightList, Graphics g)

{

AlgoritemHelper.LensAlgoritem(lightList, g, this);

}

}

}

מחלקה: Source

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Drawing.Drawing2D;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Runtime.InteropServices;

namespace Optical\_Simulation

{

[Serializable]

public abstract class Source:PhysicalObject,ISource

{

protected int numberOfSources = 1;

protected int maxSources = 200;

public Source()

{

this.IsOn = true;

this.NumberOfSources = 1;

this.Height = 60;

this.Width = 60;

this.DefaultColor = Color.Yellow;

this.FocusColor = Color.Gold;

this.LightColor = Color.Black;

}

public override PointF[] GetPoints()

{

PointF[] points = new PointF[1];

points[0] = new PointF(this.X, this.Y);

return points;

}

public override Region GetRegion()

{

return new Region(this.GetPath());

}

public override GraphicsPath GetPath()

{

GraphicsPath path = new GraphicsPath();

path.AddPolygon(this.GetPoints());

return path;

}

[Category("Source"), Description("If the source is turened on or off")]

public bool IsOn { set; get; }

[Category("Source"), Description("Number of sources")]

public int NumberOfSources

{

set { if (value >= 0 && value <= maxSources)this.numberOfSources = value; else UserValuesWarning(0, maxSources); }

get { return this.numberOfSources; }

}

[Category("Source"), Description("Default Color of light coming from the source")]

public Color LightColor { set; get; }

public override void Draw(Graphics g,bool highlighted)

{

Region reg = this.GetRegion();

SolidBrush brushDefault = new SolidBrush(this.DefaultColor);

SolidBrush brushFocus = new SolidBrush(this.FocusColor);

if (highlighted) g.FillRegion(brushFocus, reg);

else g.FillRegion(brushDefault, reg);

g.DrawPolygon(Pens.Black, this.GetPoints());

}

public virtual List<Light> EmitLight(PointF start, double angle)

{

List<Light> ray = new List<Light>();

Light light = new Light(start, MathHelper.GetEndLight(start, angle));

light.LightColor = this.LightColor;

ray.Add(light);

return ray;

}

public override String ToString()

{

return "";

}

}

}

מחלקה: StraightSource

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Drawing.Drawing2D;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Runtime.InteropServices;

namespace Optical\_Simulation

{

[Serializable]

class StraightSource:Source

{

public StraightSource()

{

}

public override object GetCopy()

{

StraightSource obj = new StraightSource();

obj.Angle = this.angle;

obj.defaultColor = this.defaultColor;

obj.focusColor = this.focusColor;

obj.height = this.height;

obj.width = this.width;

obj.x = this.x;

obj.y = this.y;

obj.numberOfSources = this.numberOfSources;

obj.IsOn = this.IsOn;

return obj;

}

public override PointF[] GetPoints()

{

PointF[] points = new PointF[4];

points[0] = new PointF(this.X - Width / 2, this.Y - Height / 4);

points[1] = new PointF(this.X - Width / 2, this.Y + Height / 4);

points[2] = new PointF(this.X + Width / 2, this.Y + Height / 2);

points[3] = new PointF(this.X + Width / 2, this.Y - Height / 2);

points = MathHelper.RotatePointFArray(this,points);

return points;

}

public PointF GetHatchPoint()

{

PointF[] p = this.GetPoints();

PointF np = new PointF();

np.X = (p[2].X + p[3].X) / 2;

np.Y = (p[2].Y + p[3].Y) / 2;

return np;

}

public override List<Light> EmitLight(PointF start,double angle)

{

List<Light> ray = new List<Light>();

Light light = new Light(start, MathHelper.GetEndLight(start, angle));

light.LightColor = this.LightColor;

ray.Add(light);

return ray;

}

public override void Draw(Graphics g,bool highlighted)

{

Region reg = this.GetRegion();

SolidBrush brushDefault = new SolidBrush(this.DefaultColor);

SolidBrush brushFocus = new SolidBrush(this.FocusColor);

if (highlighted) g.FillRegion(brushFocus, reg);

else g.FillRegion(brushDefault, reg);

g.DrawPolygon(Pens.Black, this.GetPoints());

if (this.IsOn)

{

if (this.NumberOfSources == 1)

{

PointF start = this.GetHatchPoint();

List<Light> ray = this.EmitLight(start,this.Angle);

DrawingHelper.DrawLightPath(g, ray);

}

else

{

float diffHeight = (float)MathHelper.DistanceBetweenPointF(this.GetPoints()[2], this.GetPoints()[3]) / (float)(this.NumberOfSources - 1);

for (int j = 0; j < this.NumberOfSources; j++)

{

PointF start = new PointF(this.GetPoints()[3].X + (float)(diffHeight \* j \* Math.Sin(this.Angle \* Math.PI / 180)), this.GetPoints()[3].Y + (float)(diffHeight \* j \* Math.Cos(this.Angle \* Math.PI / 180)));

List<Light> ray = this.EmitLight(start,this.Angle);

DrawingHelper.DrawLightPath(g, ray);

}

}

}

}

}

}

מחלקה: CircularSource

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Drawing.Drawing2D;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Runtime.InteropServices;

namespace Optical\_Simulation

{

[Serializable]

class CircularSource:Source

{

protected float radius,maxRadius=500;//radius of the circle which is the base from which the light come from

/// <summary>

/// Empty constructor for CircularSource, gives defult values.

/// </summary>

public CircularSource()

{

this.radius = 60;

}

public override object GetCopy()

{

CircularSource obj = new CircularSource();

obj.Angle = this.angle;

obj.defaultColor = this.defaultColor;

obj.focusColor = this.focusColor;

obj.height = this.height;

obj.width = this.width;

obj.x = this.x;

obj.y = this.y;

obj.numberOfSources = this.numberOfSources;

obj.IsOn = this.IsOn;

obj.radius = this.radius;

return obj;

}

[Category("Source"), Description("Radius of the source shape")]

public float Radius

{

set { if (value >= 0 && value <= maxRadius)this.radius = value; else UserValuesWarning(0, maxRadius); }

get { return this.radius; }

}

public override PointF[] GetPoints()

{

PointF[] points = new PointF[1];

points[0] = new PointF(this.X , this.Y);//change to Height/4

return points;

}

public override Region GetRegion()

{

return new Region(this.GetPath());

}

public override GraphicsPath GetPath()

{

GraphicsPath path = new GraphicsPath();

path.AddEllipse(this.X - (int)Radius / 2, this.Y - (int)Radius / 2, (int)Radius, (int)Radius);

return path;

}

public override void Draw(Graphics g,bool highlighted)

{

SolidBrush brushDefault = new SolidBrush(this.DefaultColor);

SolidBrush brushFocus = new SolidBrush(this.FocusColor);

if (highlighted) g.FillEllipse(brushFocus, this.X - Radius / 2, this.Y - Radius / 2, Radius, Radius);

else g.FillEllipse(brushDefault, this.X - Radius / 2, this.Y - Radius / 2, Radius, Radius);

g.DrawEllipse(Pens.Black, this.X - Radius / 2, this.Y - Radius / 2, Radius, Radius);

if(this.IsOn)

{

if (this.NumberOfSources == 1)

{

PointF start = this.CenterPoint;

List<Light> ray = this.EmitLight(start,this.Angle);

DrawingHelper.DrawLightPath(g, ray);

}

else

{

double diff = (360.0 / this.NumberOfSources);

for (int j = 0; j < this.NumberOfSources; j++)

{

PointF start = this.CenterPoint;

List<Light> ray = this.EmitLight(start, (diff \* j + this.Angle) % 360);

DrawingHelper.DrawLightPath(g, ray);

}

}

}

}

}

}

מחלקה: Mirror

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Drawing.Drawing2D;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Runtime.InteropServices;

namespace Optical\_Simulation

{

[Serializable]

class Mirror : PhysicalObject, IInteract

{

//constructor

public Mirror()

{

//defult values

this.DefaultColor = Color.Aquamarine;

this.FocusColor = Color.Aqua;

this.Height = 100;

this.Width = 30;

this.ShowImage = false;

this.ImageColor = Color.Blue;

}

public override object GetCopy()

{

Mirror obj = new Mirror();

obj.Angle = this.angle;

obj.defaultColor = this.defaultColor;

obj.focusColor = this.focusColor;

obj.height = this.height;

obj.width = this.width;

obj.x = this.x;

obj.y = this.y;

return obj;

}

public override PointF[] GetPoints()

{

PointF[] points = new PointF[4];

points[0] = new PointF(this.X - Width / 2, this.Y - Height / 2);

points[1] = new PointF(this.X - Width / 2, this.Y + Height / 2);

points[2] = new PointF(this.X + Width / 2, this.Y + Height / 2);

points[3] = new PointF(this.X + Width / 2, this.Y - Height / 2);

points = MathHelper.RotatePointFArray(this, points);

return points;

}

public override Region GetRegion()

{

return new Region(this.GetPath());

}

public override GraphicsPath GetPath()

{

GraphicsPath path = new GraphicsPath();

path.AddPolygon(this.GetPoints());

return path;

}

[Category("Mirror"), Description("Determine whether or not to show the image created by the mirror or not")]

public bool ShowImage { set; get; }

[Category("Mirror"), Description("Determine the color of the image created by the mirror")]

public Color ImageColor { set; get; }

public bool IsIntersect(Light ray, Graphics g)

{

Region ObjReg = this.GetRegion();

Region rayReg = ray.GetRegion();

ObjReg.Intersect(rayReg);

if (!ObjReg.IsEmpty(g)) return true;

return false;

}

public void Interact(List<Light> lightList, Graphics g)

{

AlgoritemHelper.MirrorAlgoritem(lightList, g, this);

}

public override void Draw(Graphics g, bool highlighted)

{

Region reg = this.GetRegion();

SolidBrush brushDefault = new SolidBrush(this.DefaultColor);

SolidBrush brushFocus = new SolidBrush(this.FocusColor);

if (highlighted) g.FillRegion(brushFocus, reg);

else g.FillRegion(brushDefault, reg);

g.DrawPolygon(Pens.Black, this.GetPoints());

}

}

}

מחלקה: Medium

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Drawing.Drawing2D;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Runtime.InteropServices;

namespace Optical\_Simulation

{

[Serializable]

class Medium : PhysicalObject, IInteract

{

protected double index = 1.5,maxIndex=10;//index of reflection

public Medium()

{

this.DefaultColor = Color.Transparent;

this.FocusColor = Color.Transparent;

this.Height = 150;

this.Width = 50;

this.Index = 1.5;

}

public override object GetCopy()

{

Medium obj = new Medium();

obj.Angle = this.angle;

obj.defaultColor = this.defaultColor;

obj.focusColor = this.focusColor;

obj.height = this.height;

obj.width = this.width;

obj.x = this.x;

obj.y = this.y;

return obj;

}

[Category("Medium"), Description("Index of the medium. Determine the curvature of light")]

public double Index

{

set { if (value >= 0 && value <= maxIndex)this.index = value; else UserValuesWarning(0, maxIndex); }

get{return this.index;}

}

public override PointF[] GetPoints()

{

PointF[] points = new PointF[2];

points[0] = new PointF(this.X-this.height/4, this.Y + this.Height / 2);

points[1] = new PointF(this.X - this.height / 4, this.Y - this.Height / 2);

points = MathHelper.RotatePointFArray(this, points);

return points;

}

public override Region GetRegion()

{

Light light = new Light(this.GetPoints()[0], this.GetPoints()[1]);

return light.GetRegion();

}

public override GraphicsPath GetPath()

{

GraphicsPath path = new GraphicsPath();

PointF[] p = this.GetPoints();

path.AddPie(this.X - this.Height / 2 - this.height / 4, this.Y - this.Height / 2, this.Height, this.Height, 360 - (float)this.angle - 90, 180);

return path;

}

public bool IsIntersect(Light ray, Graphics g)

{

Region ObjReg = this.GetRegion();

Region rayReg = ray.GetRegion();

ObjReg.Intersect(rayReg);

if (!ObjReg.IsEmpty(g)) return true;

return false;

}

public void Interact(List<Light> lightList, Graphics g)

{

Light last = lightList[lightList.Count - 1];

PointF pi = last.pi;

PointF pf = last.pf;

Color lightColor = last.LightColor;

Region mediumReg = this.GetRegion();

Region rayReg = last.GetRegion();

double angle = last.GetAngle();

mediumReg.Intersect(rayReg); //Intersect area of lens and the last light

if (!mediumReg.IsEmpty(g))

{

double n1 = ScreenProp.screenIndex;

double n2 = this.Index;

if (n1 == n2) //act as nothing

{

//

}

else

{

AlgoritemHelper.MediumAlgoritem(lightList, g, this, n1, n2);

}

}

}

public override void Draw(Graphics g, bool highlighted)

{

Region reg = this.GetRegion();

SolidBrush brushDefault = new SolidBrush(this.DefaultColor);

SolidBrush brushFocus = new SolidBrush(this.FocusColor);

if (highlighted) g.FillRegion(brushFocus, reg);

else g.FillRegion(brushDefault, reg);

g.DrawPath(Pens.Black,this.GetPath());

//g.DrawArc(Pens.Black,new RectangleF(this.X,this.Y,this.Width,this.Height),90,180);

}

[Browsable(false)]

public override float Width

{

get

{

return base.Width;

}

set

{

base.Width = value;

}

}

}

}

מחלקה: Block

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Drawing.Drawing2D;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Runtime.InteropServices;

namespace Optical\_Simulation

{

[Serializable]

class Block : PhysicalObject, IInteract

{

/// <summary>

/// Empty constructor for Block, gives defult values.

/// </summary>

public Block()

{

this.DefaultColor = Color.Black;

this.FocusColor = Color.Black;

this.Height = 100;

this.Width = 30;

}

public override object GetCopy()

{

Block obj = new Block();

obj.Angle = this.angle;

obj.defaultColor = this.defaultColor;

obj.focusColor = this.focusColor;

obj.height = this.height;

obj.width = this.width;

obj.x = this.x;

obj.y = this.y;

return obj;

}

public override PointF[] GetPoints()

{

PointF[] points = new PointF[4];

points[0] = new PointF(this.X - Width / 2, this.Y - Height / 2);

points[1] = new PointF(this.X - Width / 2, this.Y + Height / 2);

points[2] = new PointF(this.X + Width / 2, this.Y + Height / 2);

points[3] = new PointF(this.X + Width / 2, this.Y - Height / 2);

points = MathHelper.RotatePointFArray(this, points);

return points;

}

public override Region GetRegion()

{

return new Region(this.GetPath());

}

public override GraphicsPath GetPath()

{

GraphicsPath path = new GraphicsPath();

path.AddPolygon(this.GetPoints());

return path;

}

public bool IsIntersect(Light ray, Graphics g)

{

Region ObjReg = this.GetRegion();

Region rayReg = ray.GetRegion();

ObjReg.Intersect(rayReg);

if (!ObjReg.IsEmpty(g)) return true;

return false;

}

public void Interact(List<Light> lightList, Graphics g)

{

AlgoritemHelper.BlockAlgoritem(lightList, g, this);

}

public override void Draw(Graphics g, bool highlighted)

{

Region reg = this.GetRegion();

SolidBrush brushDefault = new SolidBrush(this.DefaultColor);

SolidBrush brushFocus = new SolidBrush(this.FocusColor);

if (highlighted) g.FillRegion(brushFocus, reg);

else g.FillRegion(brushDefault, reg);

g.DrawPolygon(Pens.Black, this.GetPoints());

}

}

}

חלק ממחלקה: Main\_Form

// ----- Main Receiving Function -----

// This is the main function, which handle the events after the user click on the Display

// The function, depending on the selected picturebox, will create a new object (lens, source or other)

// while creating the new object it will give it some initial data depending on which object is it

// such as the curser loctation on the display and the object type

private void Display\_MouseClick(object sender, MouseEventArgs e)

{

try

{

if (objectsList.Count < maxObjects && selectedPic != -1)

{

// setting new point, which is where the cursor is on the display

Point currentLocation = this.displayScreen.PointToClient(Cursor.Position);

//creating objects corresponding to the dropped picturebox

if (selectedPic == 1 || selectedPic == 2 || selectedPic == 3 || selectedPic == 4 || selectedPic == 5)

{

objectsList.Add(new Lens(selectedPic));

}

else if (selectedPic == 6)

{

objectsList.Add(new StraightSource());

}

else if (selectedPic == 7)

{

objectsList.Add(new CircularSource());

}

else if (selectedPic == 8)

{

objectsList.Add(new Block());

}

else if (selectedPic == 9)

{

objectsList.Add(new Medium());

}

else if (selectedPic == 10)

{

objectsList.Add(new Mirror());

}

PhysicalObject ob = objectsList[objectsList.Count - 1];

ob.X = currentLocation.X;

ob.Y = currentLocation.Y;

highlightedIndex = objectsList.Count - 1;

SaveObjectsState();

propTab.SelectedIndex =0;

UpdateInfo();

selectedPic = -1;

}

else if (objectsList.Count >= maxObjects && selectedPic != -1)

{

MessageBox.Show("Too many obejcts on screen");

}

selectedPic = -1;

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

// ----- Main Graphics Function -----

private void Display\_Paint(object sender, PaintEventArgs e)

{

Graphics g = e.Graphics;

try

{

// Loop draw every object (polymorphism)

for (int i = 0; i < objectsList.Count; i++)

{

objectsList[i].Draw(g, i == highlightedIndex);

}

//

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

// ----- Main Timer Function -----

// This is the timer which update the display, therefore is need to be used whenever the user making changes

private void timerMove\_Tick(object sender, EventArgs e)

{

try

{

//update values

displayWidth = displayScreen.Width;

displayHeight = displayScreen.Height;

timerMove.Interval = ScreenProp.screenSpeed;

//If the mouse is down on the display, the display will be updated

if (isMouseDown)

{

int index = MathHelper.DetectCollision(mouseDownLocation);//object index or -1 if none

if (index != -1)

{

mouseDownLocation = this.displayScreen.PointToClient(Cursor.Position);

if (mouseDownLocation.X < 0 || mouseDownLocation.X > displayScreen.Width || mouseDownLocation.Y < 0 || mouseDownLocation.Y > displayScreen.Height)

{

DeleteObject(index);//if the object get out of the display it will be deleted

}

else

{

//update object info

objectsList[index].X = mouseDownLocation.X;

objectsList[index].Y = mouseDownLocation.Y;

highlightedIndex = index;

}

propTab.SelectedIndex = 0;//set the propTab to the object

UpdateInfo();

}

}

else

{

SaveObjectsState();

timerMove.Enabled = false;

}

}

catch { }

}

1. קישור: https://www.visualstudio.com/ [↑](#footnote-ref-1)
2. קישור: https://netbeans.org/ [↑](#footnote-ref-2)
3. קישור: https://www.opengl.org/ [↑](#footnote-ref-3)
4. קישור: https://msdn.microsoft.com/en-us/library/system.drawing.graphics(v=vs.110).aspx [↑](#footnote-ref-4)
5. " Rotate a point around another point ",2013 ,אתר: stackoverflow, תאריך שליפה: 20.2.16, כתובת: http://stackoverflow.com/questions/13695317/rotate-a-point-around-another-point [↑](#footnote-ref-5)
6. אתר: phet.colorado.edu, תאריך שליפה: 20.2.16, כתובת: https://phet.colorado.edu/sims/geometric-optics/geometric-optics\_en.html [↑](#footnote-ref-6)